VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NALEZENÍ SLOVNÍCH KOŘENŮ V ČEŠTINĚ

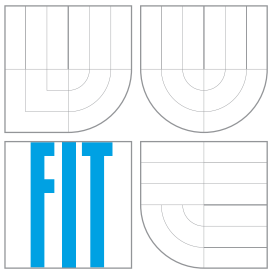
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID HELLEBRAND

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NALEZENÍ SLOVNÍCH KOŘENŮ V ČEŠTINĚ

STEMMING OF CZECH WORDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID HELLEBRAND

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2010

Zadání a licenční smlouva jsou uvedeny v archivním výtisku uloženém v knihovně FIT VUT v Brně.

Abstrakt

Cílem této diplomové práce je vytvořit pro český jazyk lemmatizační algoritmus založený na gramatických pravidlech. Práce obsahuje popis problematiky lemmatizace a několika různých lemmatizačních algoritmů. Dále jsou popsány základy gramatiky českého jazyka a také jazyka Snowball, ve kterém budou navržené postupy implementovány. Hlavní část tvoří popis implementace lemmatizačního algoritmu.

Abstract

The goal of this master's thesis is to develop stemming algorithm for czech language based on grammatical rules. You can find a description of stemming process and a comparsion of stemming algorithms in this project. The basics of czech grammar and Snowball language are also described here. The main part of this thesis concerns the implementation of the new czech stemming algorithm.

Klíčová slova

Lemmatizace, lemmatizační algoritmy, lemmatizátor, Snowball, český jazyk, gramatická pravidla.

Keywords

Stemming, stemming algorithms, stemmer, Snowball, czech language, grammatical rules.

Citace

David Hellebrand: Nalezení slovních kořenů v češtině, diplomová práce, Brno, FIT VUT v Brně, 2010

Nalezení slovních kořenů v češtině

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Hellebrand
25. května 2010

Poděkování

Děkuji svému vedoucímu Ing. Petru Chmelařovi za odborné vedení a podněty, které mi při řešení tohoto projektu poskytl.

Obsah

Kapitola 1

Úvod

Lemmatizace je proces určování základního tvaru slova (tzv. *stem*), přičemž výsledný stem se nemusí nutně shodovat s morfologickým kořenem slova. V českém jazyce, který má bohaté skloňování slov, se jedná o často užívanou metodu. Protože však je nalezení kořene slova v češtině algoritmicky složité, může být dostačující nalezení takové části slova, která bude stejná pro všechna příbuzná slova.

Lemmatizační algoritmy mohou být využity zejména v oblastech získávání informací (*information retrieval*). Typickým příkladem jsou např. webové vyhledávače. U vyhledávačů může mít lemmatizace ovšem za určitých podmínek také negativní dopad, protože rozšiřuje množinu hledaných slov a tak se ve výsledcích vyhledávání mohou objevit i nežádoucí dokumenty. Dalším způsobem využití může být např. sestavení tezauru (slovníku synonym), případně jiných produktů využívajících zpracování přirozeného jazyka.

Z historického hlediska se lemmatizátory dělily buď na slovníkové, nebo algoritmické. V současnosti ale nejsou tyto dva přístupy tak přísně rozděleny. Algoritmický lemmatizátor může obsahovat dlouhé seznamy výjimek, které můžeme chápat jako účinné slovníky. Stejně tak i slovníkový lemmatizátor potřebuje umět odstraňovat alespoň některé přípony, aby vůbec bylo možné daná slova ve slovníku vyhledávat.

Cílem této prace je implementovat v jazyce Snowball lemmatizátor českého jazyka založený na gramatických pravidlech, který nemusí nutně umět najít přesný morfologický kořen slova, ale měl by všechny existující tvary slova převádět na stejný tvar (tzv stem).

Druhá kapitola pojednává o procesu získávání informací a o vlivu předzpracování dokumentů na výkon celého procesu. Jednou z fází předzpracování dokumentů je mj. také lemmatizace.

Tématem třetí kapitoly je detailní popis procesu lemmatizace. Popisuji zde základní informace o fungování lemmatizace, o historických přístupech k tvorbě prvních lemmatizátorů a také o principu několika různých lemmatizačních algoritmů.

Ve čtvrté kapitole se zabývám českým jazykem. Je zde popsán vývoj českého jazyka a vliv historických událostí na podobu češtiny. Dále následuje popis jednotlivých českých nářečí a také základy gramatiky českého jazyka. Podrobně je rozebírána morfologie a slovotvorba v češtině.

Pátá kapitola je věnována jazyku Snowball, ve kterém bude vytvářen lemmatizátor pro český jazyk. Popisuji zde, jakým způsobem je možné Snowball zprovoznit, jaké jsou základní principy fungování tohoto jazyka a dále základní sadu příkazů umožňující efektivní práci s jazykem.

Kapitola šestá shrnuje požadavky kladené na lemmatizátor češtiny, zkoumá, jaké existují možnosti uplatnění otevřených slovníků češtiny při vývoji lemmatizátoru a popisuje, jakým

způsobem bude probíhat testování výsledků.

Sedmá kapitola pojednává o implementaci lemmatizátoru češtiny. Je zde ukázáno, jaké kroky je zapotřebí podniknout při přípravě nového jazykového modulu. Dále se věnuji vysvětlení základní funkční struktury programu a popisu průběhu vývoje jednotlivých částí lemmatizátoru. Na závěr je představena struktura kompletní verze mého lemmatizátoru pro češtinu a testování dosažených výsledků.

Závěrečná kapitola shrnuje požadavky kladené na výslednou aplikaci a shrnuje dosažené výsledky. Dále jsou uvedena opatření, která by mohla vést ke zlepšení výsledků a také je představeno, na jakém principu by mohl fungovat ideální lemmatizátor pro český jazyk.

Tato diplomová práce přímo navazuje na výsledky semestrálního projektu, v němž byla zpracována teoretická část práce. Konkrétně se jedná o kapitoly č. 3 a 4.

Kapitola 2

Získávání informací

Získáváním informací (*information retrieval*) je myšleno jednak vyhledávání dokumentů, informací v dokumentech a metadat dokumentů, ale také vyhledávání v relačních databázích a na internetu.

Ne všechna slova v dokumentu mají z hlediska sémantiky stejný význam. V psaných textech nesou některá slova více informací než jiná. Z hlediska obsahu dokumentu bývají nejdůležitější podstatná jména. Z tohoto důvodu je zpravidla vhodné provést předzpracování textu dokumentu a rozhodnout, která slova (*terms*) lze v dokumentu považovat za klíčová slova (*index terms*). Toto předzpracování textu se může skládat z několika operací, jako např. odstranění stopslov, lemmatizace, tvorba tezauru a komprese textu. Informace v této kapitole pocházejí z [1].

2.1 Základní informace

Snaha popsat dokument množinou klíčových slov má za následek nepřesné vyjádření jeho sémantiky. Např. zájmeno „ten“ samo o sobě nemá žádný význam a pokus o jeho využití pro vyhledání může vést k získání mnoha různých dokumentů, které však vůbec nijak nesouvisí s uživatelovým dotazem. Použití všech slov v dokumentu pro indexování jeho obsahu má tedy za následek generování velkého množství šumu (*noise*) ve výsledcích vyhledávání.

Jednou z možností, jak redukovat množství nepřesných výsledků, je omezit množinu slov, která budou následně použita pro popis obsahu (indexování) dokumentu. Na předzpracování dokumentu tedy můžeme zjednodušeně pohlížet jako na proces, který reguluje velikost množiny klíčových slov použitých pro popis obsahu dokumentu. Použití této omezené množiny klíčových slov by navíc mělo vést ke zlepšení výkonu vyhledávacího procesu. K dalšímu zlepšení výkonu mohou kromě předzpracování dokumentu vést i jiné typy operací. Patří k nim např. tvorba tezauru reprezentující vztahy mezi slovy a shlukování souvisejících dokumentů.

Normalizace textu a vytváření tezauru jsou operace zaměřené na zvýšení přesnosti poskytovaných výsledků. Avšak v dnešní době obrovských digitálních knihoven se dalším podstatným měřítkem stává také rychlost. Vhodným prostředkem pro zrychlení zpracování dotazů může být komprese textu. Dobrý kompresní algoritmus je schopný zredukovat text na 30 % jeho původní velikosti. Díky tomu zabírá komprimovaný text méně místa na záznamových médiích a může být rychleji přenášen přes komunikační linky. Hlavní nevýhodou je doba potřebná pro kompresi a dekompresi textu.

Až donedávna převažoval názor, že komprimování textu v podstatě nepřinese žádné

výhody, protože doba potřebná pro kompresi a dekompresi textu je delší než zrychlení způsobené použitím komprimovaného textu. Nové způsoby komprese však tento pohled na věc pomalu mění. Moderními kompresními technikami rozumíme takové, které umí rychle provádět kompresi a dekompresi, umožňují rychlý náhodný přístup bez nutnosti dekódovat komprimovaný text od jeho začátku a zvládají vyhledávání přímo v komprimovaném textu, bez nutnosti jej dekomprimovat.

2.2 Předzpracování dokumentu

Předzpracování dokumentu je podle [1] možné rozdělit na pět hlavních operací s textem:

1. Lexikální analýza textu, jejímž účelem je vypořádat se s čísly, pomlčkami, interpunkcí a rozdílnou velikostí písmen.

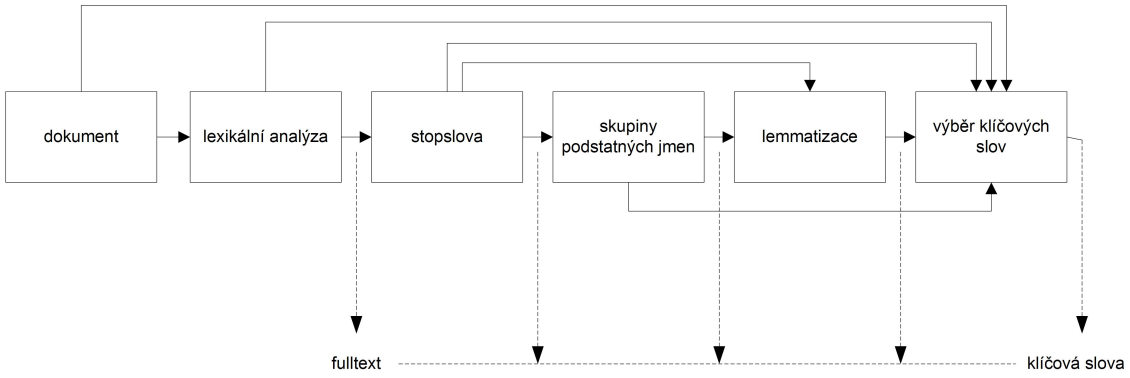
2. Odstranění stopslov, které z textu odfiltruje slova s malou informační hodnotou pro získávání informací.

3. Lemmatizace zbývajících slov zaměřená na odstranění předpon a přípon a umožňující tak získávání informací z dokumentů obsahujících různé syntaktické varianty hledaných slov (např. spojení, spojené, spojovat atd.).

4. Výběr klíčových slov. Obvykle toto rozhodnutí závisí na slovním druhu daného slova, protože např. podstatná jména nesou obvykle více informace než přídavná jména a slovesa.

5. Vytvoření kategorizačních struktur (např. tezaurus), které umožňují rozšíření původního dotazu o související slova.

Obrázek 2.1 ukazuje, jak je dokument v průběhu jednotlivých fází předzpracování převáděn z textu na množinu klíčových slov.



Obrázek 2.1: Podoba dokumentu během jednotlivých fází předzpracování [1].

Lexikální analýza

V [1] je uvedeno, že lexikální analýza je proces, během něhož se mění proud znaků (text dokumentu) na proud slov (potenciální klíčová slova dokumentu). Jedním z hlavních úkolů

lexikální analýzy je tedy identifikace slov v textu. Na první pohled se může zdát, že k rozdělení na slova stačí rozpoznávat v textu mezery (a případně více mezer za sebou redukovat na jedinou). Nicméně není to tak jednoduché. V úvahu se musí brát také číslice, pomlčky, interpunkce a velikost písmen.

Čísla se obvykle neřadí mezi klíčová slova, protože bez kontextu není jejich význam jasný. Uvažujme, že uživatel bude mít zájem o dokumenty o počtu úmrtí při automobilových nehodách mezi lety 1910 a 1989. Takovýto požadavek by mohl být charakterizován množinou klíčových slov {úmrtí, automobil, nehody, roky, 1910, 1989}. Avšak přítomnost čísel 1910 a 1989 v klíčových slovech bude mít za následek, že bude nalezeno také velké množství dokumentů vztahujících se k jednomu z těchto roků. Na základě uvedené skutečnosti by se dobrým řešením mohlo jevit nebrat v úvahu čísla jako klíčová slova. Mějme ale např. sekvenci 16 číslic, která značí číslo kreditní karty. Takovéto číslo může být v určitém kontextu vysoce relevantní a může být považováno za klíčové slovo. Předběžným přístupem, jak se vypořádat s čísly v textu, může být odstranění všech slov obsahujících sekvenci číslic, jestliže nejsou specifikovány jiným způsobem (např. regulárním výrazem). Dále pak může pokročilý způsob lexikální analýzy zajistit normalizaci dat a čísel do jednotného formátu.

Zpracování pomlček představuje další obtížnou část činnosti lexikálního analyzátoru. Vhodnou možností je rozdělení slov spojených pomlčkou. Dále bývají v průběhu lexikální analýzy obvykle odstraněna interpunkční znaménka.

Velikost písmen zpravidla nemá vliv na nalezení klíčových slov dokumentu. Lexikální analyzátor převádí veškerý text buď na malá, nebo na velká písmena. U některých slov však může změna velikosti písmen způsobit změnu významu daného slova.

Všechny výše uvedené operace mohou být provedeny snadno. Nicméně u každé z nich je potřebné postupovat uvážlivě, protože mohou mít významný vliv na dobu potřebnou pro zpracování dokumentu.

Odstranění stopslov

Podle [1] není vhodné slova vyskytující se v dokumentu příliš často používat jako klíčová slova. Pokud se totiž některé slovo nachází v 80 % dokumentů, je pro účely získávání informací naprosto bezcenné. Takováto slova se označují jako stopslova a vylučují se z dalšího zpracování dokumentu. Nejčastěji se jedná o předložky, spojky a částice.

V některých případech získávání informací mohou být stopslova také užitečná. Příkladem může být hledání ve frázích („být či nebýt“ apod.). V uvedeném příkladě by po odstranění stopslov prakticky nebylo možné nalézt dokumenty, které obsahují hledanou frázi.

Odstranění stopslov přináší jednu podstatnou výhodu - významně redukuje velikost zpracovávaného textu. Pouhým odstraněním stopslov je možné zmenšit objem textu o 40 a více procent.

Lemmatizace

Problematika lemmatizace je podrobně rozebrána v kapitole 3.

Výběr klíčových slov

Pro získávání informací z dokumentu může dle [1] sloužit buď celý jeho obsah (všechna slova v textu), nebo může být daný dokument popsán pouze určitou charakteristickou

množinou slov. Tato tzv. klíčová slova musí být určitým způsobem vhodně zvolena. Mohou být vybrána člověkem (specialistou), případně může být výběr proveden automatizovaně.

Existuje více možných postupů pro automatický výběr klíčových slov v dokumentu. Jedním z nich je hledání skupin podstatných jmen (*identification of noun groups*). Nejvíce informace v textu nesou právě podstatná jména, proto je vhodné použít je jako klíčová slova dokumentu. Místo jediného podstatného jména se jako klíčové slovo použije celá skupina podstatných jmen. Skupina se skládá z takových podstatných jmen, která mají v textu od sebe určitou vzdálenost. To znamená, že počet slov mezi dvěma podstatnými jmény nepřekročí určitý zadaný práh (např. 3).

Tezaurus

V [1] je uvedeno, že slovo tezaurus pochází z latinského „thesaurus“ a znamená poklad, či pokladnice. V nejjednodušší formě se tezaurus skládá ze seznamu slov a ke každému slovu z tohoto seznamu dále obsahuje jeho synonyma.

Ve většině případů však není tezaurus pouhým seznamem slov a jejich synonym, ale obsahuje mnohem komplexnější struktury zachycující vztahy mezi slovy. K hlavním účelům tezauru patří:

- Poskytnutí slovní zásoby pro indexaci dokumentů a vyhledávání v nich.
- Pomoc uživateli se sestavením vhodného dotazu.
- Poskytnutí setříděné hierarchické struktury umožňující uživateli upřesnit jeho dotaz tak, jak potřebuje.

Kapitola 3

Lemmatizace

Lemmatizace (stemming) je postup, během kterého se slova převádějí na jejich základní tvar – tzv. stem. Výsledný stem se nemusí nutně shodovat s morfologickým kořenem slova. Obvykle je dostačující, pokud jsou příbuzná slova převedena na stejný tvar, i když tento tvar není přesným kořenem slova.

Např. výsledkem lemmatizace u slov „kance“, „kanci“ a „kancem“ by v ideálním případě mělo být slovo „kan“, které je morfologickým kořenem všech tří slov. Pokud by se nepodařilo najít přímo kořen, měl by lemmatizátor vrátit alespoň nejdelší společnou část slov, tedy např. „kanc“.

Programy provádějící lemmatizaci bývají souhrnně označovány jako lemmatizační algoritmy (*stemming algorithm*) nebo jako lemmatizátory (*stemmer*).

3.1 Základní informace

Základní myšlenkou lemmatizace je podle [9] snaha o vylepšení procesu získávání informací (*information retrieval*) tím, že k několika slovům majícím podobný význam určí jejich společný základ, ze kterého byly odvozeny.

Lemmatizaci není možné provádět s úplně všemi jazyky. Například s čínštinou si neporadí. Naproti tomu u jazyků patřících do Indo-Evropské rodiny jazyků jsou jednotlivá slova tvořena na základě různých gramatických pravidel, takže je možné na ně použít některý lemmatizační algoritmus.

Většina evropských a mnoho asijských jazyků patří do Indo-Evropské rodiny jazyků. Historicky zahrnuje tato rodina také starověkou latinu, řečtinu a perštinu. Díky rozmachu evropských říší nyní tyto jazyky dominují také v Americe, Austrálii a velké části Afriky. Indo-Evropské jazyky jsou tedy hlavními jazyky celého moderního západního světa a u všech je možné provádět lemmatizaci.

Indo-Evropskou rodinu jazyků je dále možné rozdělit na menší skupiny jazyků, např. románské (italština, francouzština, španělština . . .), slovanské (ruština, polština, čeština . . .), germánské (němčina, holandština . . .) atd.

Jako *stem* budeme označovat výsledek procesu lemmatizačních algoritmů. Naproti tomu *kořen slova* budeme chápat opravdu jako gramatický slovní kořen. Tedy jako základní tvar daného slova.

Pro rodinu Indo-Evropských jazyků je typické, že základem každého slova je kořen slova. Před ním bývají umístěny předpony (*prefixy*), za ním přípony (*sufixy*). Předpony i přípony mají souhrnný název *afixy*.

V [9] je uvedeno, že přípony můžeme rozdělit do tří základních skupin: D-, I- a A-přípony.

A-přípona (*attached suffix*) je slovo připojené k jinému slovu. Tento typ přípon se často objevuje v italštině, španělštině a také v portugalštině (ačkoliv v portugalštině bývá od předcházejícího slova oddělen pomlčkou, takže je snadné je odstranit).

Např. v italštině se osobní zájmeno připojuje ke slovesu:

- mandargli = mandare + gli = poslat + jemu

I-přípona (*inflectional suffix*) vychází ze základních pravidel gramatiky daného jazyka. Je možné ji použít na všechna slova určitého slovního druhu (i když může existovat malé množství výjimek). Např. v angličtině se minulý čas sloves tvoří přidáním přípony „ed“. V některých případech je nutné navíc upravit tvar kořene slova.

- fit + ed = fitted (zdvojené „t“)
- love + ed = loved (vypuštění koncového „e“ ve slově „love“)

V případě I-přípon je také poměrně běžné, že mohou mít více funkcí. Např. v angličtině může přípona „s“ znamenat:

- Sloveso ve třetí osobě jednotného čísla (runs, sings)
- Množné číslo podstatného jména (dogs, cats)
- Podstatné jméno označující vlastnictví (boy's, girl's)

Ale jinak je možné toto pravidlo použít na všechna slovesa současné angličtiny, kromě zhruba 150 nepravidelných sloves (např. become – became, begin – began, atd.).

D-přípona (*derivational suffix*) umožňuje z jednoho slova vytvořit slovo jiné, častokrát jiného slovního druhu nebo jiného významu. Může-li být určitá D-přípona přidána k nějakému slovu, nelze zjistit na základě pravidel gramatiky daného jazyka, ale pouze ve spolupráci se slovníkem. Např. v angličtině může přidání přípony „ness“ k určitým přídavným jménům vést ke vzniku odpovídajících podstatných jmen (kindness, foolishness . . .). Toto však není možné provádět se všemi přídavnými jmény (nelze např. u slov big, cruel . . .).

D-přípona může mít více než jednu funkci. Např. v angličtině přípona „ly“ vytvoří z přídavného jména příslovce (greatly), ale také může z podstatného jméno vytvořit přídavné (kingly). Ve francouzštině může přidáním přípony „ement“ vzniknout z přídavného jméno příslovce (grandement), ale také ze slovesa podstatné jméno (rapprochement).

Vzhledem k tomu, že ve slově za sebou přípony zpravidla následují v pořadí D, I a A, budou zpravidla odstraňovány zprava v pořadí A, I a D. Obvykle se snažíme o odstranění všech A- a I-přípon a alespoň některých D-přípon.

3.2 Historie lemmatizace

Na nalezení algoritmu, který by uměl správně určovat základní tvary slov, se pracuje už několik desetiletí. Autorem prvního lemmatizátoru je Julie Beth Lovins. Její práce představující tento lemmatizátor byla publikována roku 1968 v článku [7] a v následujících letech významně ovlivnila vývoj v oblasti lemmatizace.

Jeden z dalších zásadních lemmatizátorů publikoval Martin F. Porter v roce 1980 v článku [10]. Tento lemmatizátor začal být hojně používaným a v podstatě se stal základním algoritmem pro lemmatizaci anglického jazyka.

V [12] je uvedeno, že se v průběhu let objevovaly mnohé implementace Porterova algoritmu, avšak velké množství těchto implementací obsahovalo větší i menší chyby, což mělo za následek, že tyto lemmatizátory nemohly plně využívat svůj potenciál. Aby Porter zamezil vznikání dalších chybných implementací, vydal v roce 2000 volně šiřitelnou implementaci jeho algoritmu. V dalších letech na základě této implementace vyvinul Snowball – framework pro tvorbu lemmatizátorů. Ve Snowballu vytvořil i vylepšenou verzi jeho algoritmu pro anglický jazyk a také algoritmy pro několik dalších jazyků.

3.3 Lemmatizační algoritmy

Existuje několik základních typů lemmatizačních algoritmů. Navzájem se od sebe liší jednak výkonností (rychlost nalezení výsledků), přesností nalezených výsledků a také schopností, jak se vypořádat s překážkami číhajícími v procesu lemmatizace daného jazyka.

3.3.1 Brute Force algoritmy

Brute Force algoritmy [12] využívá pro svou práci pouze hrubou výpočetní sílu počítače. Využívají tabulku, ve které jsou uvedeny dvojice kořen slova – vyskloňovaný tvar slova. Zjištění kořene slova probíhá tak, že se dané slovo hledá v této tabulce. Pokud je nalezeno, vrátí algoritmus příslušný kořen slova nalezený v tabulce. Ukázka tabulky Brute Force algorimu je uvedena v tabulce 3.1

odvozené slovo	**základní tvar**
tresčí	tres
tance	tan
lesník	les
lesnictví	les
...	...

Tabulka 3.1: Ukázka tabulky Brute Force algoritmů.

Stinnou stránkou Brute Force algoritmů je fakt, že neobsahují žádné postupy, které by mohly usnadnit a urychlit nalezení výsledku. Tyto algoritmy tedy během hledání výsledku provádí zbytečně větší množství operací, než by bylo nezbytně nutné. Další zápornou vlastností je potřeba relativně velkého množství paměti pro uložení tabulky s jednotlivými slovy. Tyto algoritmy mají vlastně za úkol pouze určit, jestli se dané slovo v tabulce vyskytuje nebo ne. Vzhledem k obrovskému množství slov v každém jazyce je nereálné sestavit tabulku, která bude obsahovat všechna slova a jejich různé odvozené tvary. Manuální trénování algoritmu (tzn. přidávání dalších slov do tabulky) je pro člověka velice časově náročné. Také poměr mezi úsilím vynaloženým na rozšiřování tabulky a nárůstem přesnosti výsledků algoritmu je i v nejlepším případě mizivý.

Brute Force algoritmy se však s některými nástrahami lemmatizace mohou vypořádat lépe, než jiné používané postupy. Zdaleka ne všechny vyskloňované tvary slov se v daném

jazyce řídí přesně podle gramatických pravidel. Nepravidelné tvary slov jsou toho dobrým příkladem (v češtině např. u sloves „je“, „není“, „jsem“ je základním tvarem sloveso „být“). Pro algoritmy odstraňující přípony slov (*Suffix stripping algorithms*) mohou být tato slova velkou překážkou při hledání správných výsledků, zatímco Brute Force algoritmům stačí mít v tabulce uloženou patřičnou dvojici slov a správný výsledek je na světě. Nicméně stále zůstává předpoklad, že hledaný tvar slova musí být uložen v tabulce. Pokrytí celého jazyka je tedy zásadním problémem pro vylepšování Brute Force algoritmů.

Jako alternativa k ručnímu sestavení tabulky mohou existovat pokusy o sestavení tabulky automatizovaně. V tomto případě dostane program na vstupu sadu slov daného jazyka a ke každému doplní všechny předpony a přípony daného jazyka. Takový postup samozřejmě nezajistí naplnění tabulky všemi slovy daného jazyka, ale může sloužit jako základ pro budoucí ruční doplňování tabulky. Při tomto způsobu tvoření slov ovšem dojde také k mnoha situacím, kdy spojením kořene slova s některou předponou či příponou vznikne nesmyslné slovo, které se ve skutečnosti v daném jazyce vůbec nevyskytuje. to ale nebude mít vliv na přesnost výsledků poskytovaných algoritmem používajícím takto sestavenou tabulku, neboť mezi hledanými slovy se tyto nesmyslné kombinace vyskytovat nebudou. Zásadně ovšem může být ovlivněna výkonnost algoritmu, protože tabulka nyní obsahuje značné množství zbytečných záznamů, což bude mít za následek prodloužení doby potřebné k nalezení výsledku.

Jako hlavní překážku při vývoji programu založeného na Brute Force algoritmu tedy můžeme označit nutnost uložit na počátku do tabulky obrovské množství slov, aby vůbec přesnost poskytovaných výsledků byla na přijatelné úrovni. Nicméně přesnost výsledků je možné jednoduše (ale velmi pracně) zlepšovat. Stačí do tabulky přidávat další a další slova.

3.3.2 Suffix stripping algoritmy

Suffix stripping algoritmy [12], na rozdíl od Brute Force algoritmů, nepotřebují tabulku s výčtem všech slov jazyka a k nim příslušných kořenů slov. Na místo toho jim stačí relativně malý seznam pravidel, na jejichž základě z daného slova odstraní případné přípony vyskytující se v daném jazyce. Takovýto přístup se hodí například pro anglický jazyk.

Jednoduchá ukázka pravidel pro anglický jazyk:

- pokud slovo končí na „ed“, odstraň z něj „ed“
- pokud slovo končí na „ing“, odstraň z něj „ing“
- pokud slovo končí na „ly“, odstraň z něj „ly“

Vývoj a vylepšování těchto algoritmů je v porovnání s Brute Force algoritmy mnohem jednodušší, ovšem za předpokladu, že vývojář má v požadovaném jazyce dostatečné znalosti v oblasti jazykovědy a morfologie. Problémem je u těchto algoritmů zpracování různých výjimek v jazyce. Na ně totiž obecná pravidla tvorby slov v daném jazyce neplatí a algoritmus si tedy nemá s těmito výjimkami jak poradit. Nasazení suffix stripping algoritmů je tedy omezeno pouze na slovní druhy (podstatné jméno, přídavné jméno, sloveso atd.), u kterých jsou v daném jazyce známy možné přípony a neexistuje pro ně mnoho výjimek.

U jednotlivých suffix stripping algoritmů se mohou poskytované výsledky z několika důvodů lišit:

- Prvním důvodem je skutečnost, jestli výsledný stem musí být platným slovem v daném jazyce. Některé přístupy toto nevyžadují. Jiné si zase existenci výsledného stemu

kontrolují v databázi všech platných kořenů slov daného jazyka. Pokud se v ní výsledné slovo nevyskytuje, provede algoritmus alternativní kroky (např. zkusí odstranit jinou příponu). Pro takovéto případy může mít algoritmus jednotlivým pravidlům na odstraňování přípon přiřazeny různé priority (buď zadané člověkem, nebo získané stochasticky).

- Další možností může být, že se algoritmus rozhodne neaplikovat takové pravidlo, jehož výsledkem není platné slovo daného jazyka, pokud má k dispozici nějaké jiné pravidlo dávající platný výsledek.

Vylepšením základního Suffix stripping algoritmu může být substituce přípon. Substituční pravidla určují, která přípona může být nahrazena jinou. Kvalita poskytovaných výsledků u této metody záleží především na dobrém návrhu celého algoritmu. Např. algoritmus může u daného slova detekovat, že je možné použit jak odstranění přípony, tak i její nahrazení jinou příponou. Protože po odstranění původní přípony by (na rozdíl od substituce) nevzniklo platné slovo, rozhodne se algoritmus raději pro nahrazení přípony. Na slovo s touto novou příponou už je následně možné odstraňovací pravidlo použít. Takže ve výsledku díky substituci přípon poskytne algoritmus platný výsledek.

Tento příklad ilustruje zásadní rozdíl mezi Brute Force a Suffix stripping algoritmy. Zatímco Brute Force by musel dané slovo hledat v obrovské databázi všech slov a jejich odvozených tvarů, Suffix stripping algoritmu stačí aplikovat na dané slovo jen několik málo pravidel a dostane se ke stejnému výsledku, ovšem pravděpodobně mnohem rychleji.

3.3.3 Lemmatizační algoritmy

Lemmatizační algoritmy (*Lemmatisation Algorithms*) [12] se snaží o komplexnější přístup k nalezení kořene slova. Nejdříve se pokouší zjistit, o jaký slovní druh se u hledaného slova jedná, a na základě toho poté aplikují pro každý slovní druh rozdílné normalizační metody a pravidla pro nalezení stemu.

Úspěch těchto algoritmů značně závisí na správném rozpoznání slovního druhu daného slova. Ačkoliv se normalizační metody a pravidla pro jednotlivé slovní druhy mohou překrývat, chybné určení slovního druhu degraduje tuto metodu na pouhý Suffix stripping algoritmus. Základní myšlenkou lemmatizačních algoritmů totiž je, že čím více a čím přesnějších informací o hledaném slově máme, tím přesněji můžeme aplikovat normalizační pravidla (která jsou více méně stejná, jako u Suffix stripping algoritmu).

3.3.4 Stochastické algoritmy

Stochastické algoritmy (*Stochastic Algorithms*) [12] využívají pro nalezení kořene slova pravděpodobnost. Nejdříve u nich probíhá tzv. trénovací fáze, během níž si ze vztahů mezi kořenem slova a z něj odvozených slov vytváří pravděpodobnostní model. Tento model zpravidla představuje množinu pravidel podobnou jako u Suffix stripping nebo lemmatizačních algoritmů. Hledání kořene slova potom probíhá tak, že natrénovanému modelu je předloženo dané slovo a model na základě své naučené množiny pravidel nalezne nejpravděpodobnější kořen daného slova.

Pokud není zcela jasné, jakého slovního druhu hledané slovo je, může algoritmus přiřadit určitou pravděpodobnost všem možnostem. V takovémto případě se algoritmus může rozhodovat podle kontextu (okolních slov ve větě), ve kterém se hledané slovo vyskytuje. V případě bezkontextových gramatik však na tyto doplňující informace není brán zřetel.

3.3.5 Analýza N-gramů

Základním principem těchto algoritmů je podle [12] rozdělení vstupního textu na N-tice po sobě jdoucích slov (případně písmen). Postupně se prochází všechna slova v textu a je vytvářena tabulka obsahující všechny N-tice nalezené ve vstupním textu. Každý řádek této tabulky N-gramů odpovídá jedné nalezené N-tici. Dále může obsahovat také některé jiné užitečné informace, jako např. četnost výskytu daného N-gramu ve vstupním textu.

Analýzou tabulky N-gramů může algoritmus pro dané slovo X zjistit, že toto slovo bude s určitou pravděpodobností následováno slovem Y.

Pokud budeme analyzovat bigramy (2-gramy), pak můžeme říct, že první slovo z dvojice určuje kontext, ve kterém budeme chápat druhé slovo z dvojice. Tedy první slovo nám poskytuje určitou doplňující informaci k významu druhého slova. Jednotlivá slova vstupního textu totiž mohou mít různý význam. Díky kontextu s okolními slovy tak může algoritmus přesněji určit správný význam slova. Čím více informací o daném slově algoritmus má, tím lépe se může rozhodnout, jak postupovat při hledání kořene slova.

Problémem u těchto algoritmů může být, pokud sestavují tabulku N-gramů z nereprezentativního vzorku textu. Tento fakt poté může zásadně ovlivnit přesnost poskytovaných výsledků.

3.3.6 Hybridní algoritmy

Hybridní algoritmy [12] kombinují několik výše uvedených přístupů. Podobně jako Brute Force algoritmy mohou nejdříve zkontrolovat, jestli se dané slovo nenachází v jejich databázi. Ovšem na rozdíl od Brute Force si v databázi neuchovávají všechna slova daného jazyka, ale jen často se vyskytující nepravidelné tvary slov. Tabulka tedy zůstává malá a je možné v ní rychle vyhledávat. Pokud se hledané slovo v tabulce nenachází, přichází na řadu další algoritmy, např. Suffix stripping.

3.4 Využití slovníků v algoritmech

V [9] je uvedeno, že z historického hlediska se lemmatizátory dělily buď na slovníkové, nebo algoritmické. V současnosti ale nejsou tyto dva přístupy tak přísně rozděleny. Algoritmický lemmatizátor může obsahovat dlouhé seznamy výjimek, které můžeme chápat jako účinné mini slovníky. Stejně tak i slovníkový lemmatizátor potřebuje umět odstraňovat alespoň I-přípony, aby vůbec bylo možné daná slova ve slovníku vyhledávat.

Mohlo by se zdát, že v porovnání se slovníkovými lemmatizátory nemohou ty algoritmické poskytovat stejně dobré výsledky, ale z následujících důvodů mohou být i přesto užitečné:

1. Algoritmické lemmatizátory mohou být velmi malé, jednoduché a rychlé.
2. I přes jimi produkované chyby jsou jejich výsledky v praxi stále dobře použitelné.
3. Slovníkové lemmatizátory vyžadují velmi mnoho úsilí pro údržbu slovníku.

3.5 Chybové metriky

Podle [9] je jedním z možných pohledů na vztah mezi termy a dokumenty, chápat dokumenty jako soustavu pojmů a termy jako slova popisující jednotlivé pojmy.

Potom ovšem může jedno slovo pokrývat více pojmů (lidské ucho - ucho u hrnce, háček jako malý hák – háček jako problém, hlavolam). V tomto příkladě jsou „ucho“ a „háček“ homonyma (slova, která stejně znějí, ale mají jiný význam). Nebo také jeden pojem může být popsán více slovy (mnoho – hodně, velmi, velice, spousta . . .). Takováto slova se nazývají synonyma (slova, která se liší svou zvukovou formou, ale mají stejný význam).

Proto tedy existuje mezi množinou termů a množinou pojmů relace N:N. Lemmatizace je proces, který díky této relaci může redukovat počet synonym, i když občas vede tato redukce ke vzniku nových homonym. V podstatě jsou tedy chyby při lemmatizaci pouze zavedení nových homonym do slovníku, který už značný počet homonym obsahuje.

V jazyce se vyskytují také slova, která nepopisují žádný pojem. Tato slova (být, skoro, jinak . . .) jsou označována jako stop-slova (*stopwords*). I stop-slova mohou být užitečná při získávání informací, ale pouze pro hledání ve frázích („být či nebýt“ atd.).

Podle [1] přináší odstranění stop-slov jednu podstatnou výhodu – významně redukuje velikost zpracovávaného textu. Odstraněním stop-slov je možné zmenšit objem textu o 40 a více procent.

V [12] je uvedeno rozdělení chybových metrik na *under-stemming* a *over-stemming*.

over-stemming znamená, že dvě různá slova dostanou algoritmem přiřazený stejný kořen slova, i když by tomu tak být nemělo – *false positive*. Jinými slovy chyba vzniklá odebráním příliš velké části přípony.

under-stemming znamená, že dvěma různým slovům by měl algoritmus přiřadit stejný kořen slova, ale není tomu tak – *false negative*. Neboli chyba vzniklá odebráním příliš malé části přípony.

Tuto metriku je možné zmenšit využitím slovníku. Pravděpodobně se nepodaří odstranit všechny chyby, ale i přesto může chybovost výrazně poklesnout.

Uvedené chyby můžeme omezit použitím slovníku. Ovšem ani slovník nám nezaručí 100% úspěšnost, ale může být nástrojem pro vylepšení poskytovaných výsledků. Samozřejmě však také záleží na kvalitě slovníku. Aby mohlo být dosaženo co možná nejlepších výsledků, musí být slovník velmi obsáhlý a často aktualizovaný.

Všechny druhy lemmatizačních algoritmů by se měly snažit minimalizovat obě chyby. Avšak omezení jedné chyby má většinou za následek nárůst druhé.

Kapitola 4

Čeština

Čeština je západoslovanský jazyk (stejně jako např. slovenština, polština, srbština). Patří do rodiny indoevropských jazyků, konkrétně mezi slovanské jazyky.

4.1 Historie

Článek [3] uvádí, že roku 863 dorazili na území Velkomoravské říše Slovanští věrozvěsti Cyril a Metoděj, kteří vedli bohoslužby ve staroslověnštině (jihoslovanský jazyk srozumitelný v té době všem Slovanům) a položili tak základy vlastní církve ve Velkomoravské říši. Sestavili hlaholici - písmo pro slovanský jazyk a překládali do staroslověnštiny liturgické texty. Toto navázání kulturního kontaktu s Byzantskou říší mělo osvobodit Velkomoravskou říši od politického a náboženského vlivu německých kmenů. Protože však misie neměla dlouhého trvání, začal brzy opět stoupat vliv Východofranské říše, takže došlo k návratu latiny a hlaholici nahradila latinka.

Ve 14. století se začínají objevovat první česky psané knihy. Karel IV. nechal vyhotovit první překlad Bible do češtiny. Na přelomu 14. a 15. století dochází k reformě pravopisu, díky které byly z češtiny odstraněny spřežky a bylo zavedeno používání diakritických znamének. Propagátorem této reformy byl Jan Hus. K velkému rozvoji česky psané literatury v 15. století významně přispěl vynález knihtisku. Jako vzor spisovného jazyka dlouho sloužila Bible kralická.

Korunovací Ferdinanda I. Habsburského na českého krále v roce 1526 začíná ovládnutí Českých zemí Habsburskou dynastií, což mělo za následek posilování pozice němčiny na úkor češtiny. Po porážce stavovského povstání v bitvě na Bílé hoře roku 1620 dochází na 200 let k úpadku českého jazyka. Česká nekatolická inteligence byla donucena k emigraci, což vedlo mj. k omezení česky psané literatury. Druhým úředním jazykem v Českých zemích se stala němčina, která tak byla zrovnoprávněna s češtinou.

K obnovení pozice češtiny postupně docházelo až v 19. století díky hnutí označovanému jako národní obrození. Dochází k pozdvižení úrovně české literatury. Díky povinné školní docházce a s ní spojené vysoké gramotnosti obyvatelstva se čeština šířila mezi všechny společenské vrstvy. V této době získala čeština víceméně svou dnešní podobu.

Dlouhé období silného vlivu němčiny, během něhož byla čeština prakticky vytlačena z veřejného života, a také skutečnost, že České země byly z velké části obklopeny německy mluvícími zeměmi, mělo samozřejmě za následek pronikání německých prvků do češtiny.

Nejbližším příbuzným jazykem češtiny je slovenština. Až do 18. století byla literatura v češtině společná pro oba národy. Od vzniku Československa v roce 1918 až do rozdělení

republiky v roce 1993 byly v médiích čeština i slovenština prezentovány rovnoměrně, takže Češi i Slováci alespoň pasivně rozuměli oběma jazykům.

V letech 1948 – 1989 vedla politická nadvláda Sovětského svazu a povinná výuka ruštiny ve školách k ovlivňování češtiny ruštinou. V dnešní době má na češtinu největší vliv anglický jazyk.

4.2 Nářečí

Nářečí češtiny jsou mluvené formy českého jazyka omezené na určité oblasti České republiky. Jednotlivé druhy nářečí jsou většinou navzájem srozumitelné.

Česká nářečí je podle [3] možné rozdělit do čtyř hlavních skupin:

- česká
- středomoravská (hanácká)
- východomoravská (moravskoslovenská)
- slezská

Území, na kterých se jednotlivá nářečí používají, jsou znázorněna na obrázku 4.1.

Většina pohraničních území nepatří do žádné z těchto 4 skupin, protože po druhé světové válce bylo tamní německé obyvatelstvo nuceně vystěhováno a nahrazeno obyvateli z různých částí tehdejšího Československa.

Na severovýchodě republiky se v oblasti u polských hranic používá mix češtiny a polštiny. Typickým rysem jsou krátké samohlásky, přízvuk na předposlední slabice a melodie řeči připomínající polštinu.

Přestože existuje velké množství různých rysů odlišujících jednotlivá nářečí, dvě největší skupiny (česká a středomoravská) je možné od sebe odlišit podle jejich rozdílného nakládání s určitými dlouhými samohláskami (viz tabulka 4.1).

česká skupina		vs.	**středomoravská skupina**	
ý →	ej		ý →	é
é →	ý/í		é →	ý/í
ú →	ou		ú →	ó
ó →	ú		ó →	ú

Tabulka 4.1: Dlouhé samohlásky v české a středomoravské skupině nářečí [3].

4.3 Fonologie

Fonologie (hláskosloví) je vědní disciplína, která studuje a popisuje zvukovou složku lidské komunikace za pomoci mluvních orgánů. Jejím cílem je zkoumání zvukové stránky přirozeného jazyka. Fonologie češtiny tedy popisuje výslovnost a funkce jednotlivých hlásek používaných v češtině [4]. Dále je v [5] uvedeno, že se fonologie zabývá těmi složkami zvukového signálu řeči, které jsou v daném jazyce významotvorné, tj. jsou schopny rozlišovat slova

Obrázek 4.1: Mapa výskytu jednotlivých skupin nářečí [4].

a slovní tvary s různou komunikační funkcí. Např. změna znělosti tvoří v češtině nové slovo: *pere – bere*. Změna intonace mění komunikační funkci, např. při poklesu hlasu na konci se jedná o konstatování (*Rozumíš.*), se stoupáním hlasu je tatáž posloupnost slabik otázkou (*Rozumíš?*). Složek zvukového signálu řeči, které jsou v daném jazyce takto relevantní, je omezený počet.

Český pravopis používá pro označení českých hlásek tato písmena: a, á, b, c, č, d, ď, e, é, ě, f, g, h, ch, i, í, j, k, l, m, n, ň, o, ó, p, r, ř, s, š, t, ť, u, ú, ů, v, y, ý, z, ž. Za jediné písmeno je pokládána spřežka ch, tj. spojení písmen c a h. Ve slovech cizího původu se dále vyskytují ještě písmena q, w, x.

Český pravopis je v zásadě hláskový [6], tzn. každá hláska se zpravidla označuje jedním zvláštním písmenem a každé písmeno zpravidla označuje jednu a tutéž hlásku.

Podle [4] patří čeština mezi jazyky s pevným přízvukem, který bývá kladen vždy na první slabiku přízvučného celku, kterým je obvykle slovo. Výjimkou jsou jednoslabičné předložky, které tvoří s následujícím jménem přízvučný celek, tedy přízvuk se přesouvá na tuto předložku a některá jednoslabičná slova (např. to, se, jsem atd.), která nemají vlastní přízvuk (jsou to tzv. příklonky) a tvoří přízvučný celek s předcházejícím slovem. Čtyř a více slabičná slova mívají navíc vedlejší přízvuk na každé liché slabice.

4.4 Morfologie

V [4] se uvádí, že morfologie (tvarosloví) je lingvistická disciplína tvořící součást klasické gramatiky. Studuje všechny typy slovních druhů z hlediska jejich formy a funkce.

Podle [5] se formou slova roumějí všechny jeho podoby – tvary, jichž se užívá ve větách. Tyto formy se skládají z tvarotvorného základu, který je společný pro všechny tvary slova, a z morfu.

Morfem rozumíme útvar nesoucí gramatický význam, a to především koncovku (*žen-y, píš-u*) včetně koncovky nulové. Morfy mohou být součástí slova, nebo jsou samostatné (volné): *bych, bys* atd.

Tvary slov se stejnými soubory morfů tvoří typ (vzor) nebo podtyp.

V [4] je dále uvedeno, že česká gramatika je založena na podobných principech jako gramatiky ostatních slovanských jazyků. Čeština je typickým příkladem flektivního jazyka. Tyto jazyky vyjadřují gramatické funkce pomocí flexe (ohýbání), tzn. skloňování, časování a používání předpon a přípon. Pracují s vázanými morfémy, takže je často těžké rozdělit slovo na jednotlivé segmenty a určit jeho kořen. Morfém je nejmenší sémanticky vydělitelná část slova, která je nositelem věcného nebo gramatického významu. Jedná se tedy o předpony, vpony, přípony, koncovky nebo kořeny slova.

Slovní druhy

V češtině se rozlišuje 10 slovních druhů: podstatná jména (substantiva), přídavná jména (adjektiva), zájmena (pronomina), číslovky (numeralia), slovesa (verba), příslovce (adverbia), předložky (prepozice), spojky (konjunkce), citoslovce (interjekce) a částice (partikule).

Podle [5] lze slovní druhy dělit na ohebné a neohebné, v rámci ohebných pak na skloňované (substantiva, adjektiva, zájmena a číslovky) a časované (slovesa). Neohebné slovní druhy jsou adverbia, předložky, spojky, citoslovce a částice. Toto dělení však není absolutní a mohou se vyskytovat výjimky.

Čeština má několik souborů skloňovacích koncovek. Jedná se o skloňování substantivní (jmenné), adjektivní (složené) a zájmenné. Substantiva se skloňují převážně jmenně (*pán, hrad*…, kromě typů *hajný, krejčí, pokojská, vedoucí, jízdné, telecí*), adjektiva převážně složeně (typy *mladý, jarní*) a také smíšeně s tvary složenými a jmennými (typy *otcův, matčin*).

Z hlediska významového (sémantického) lze pohlížet na slovní druhy jako na slova pojmenovávající různé druhy jevů reality. Dělí se na plnovýznamová a neplnovýznamová. Mezi nimi však není ostrá hranice. Za plnovýznamová se považují substantiva, adjektiva, zájmena, číslovky, slovesa a adverbia. Za neplnovýznamová předložky, spojky a částice.

4.4.1 Substantiva (podstatná jména)

V [4] se uvádí, že substantiva jsou základním slovním druhem označujícím jevy skutečnosti jako samostatná fakta. Substantiva zahrnují nejen jména osob, zvířat, věcí, jevů, ale i vlasnosti a děje pojímané jako nezávislé entity (*krása, běh*). Substantivum je ohebný slovní druh, který prostřednictvím systému tvarů vyjadřuje mluvnické významy pádu, čísla a rodu. Kromě několika málo výjimek (např. slovo „nůžky“) všechna podstatná jména existují v jednotném i v množném čísle.

Podle [5] je gramatický rod závazný pro každé jméno a zároveň je klasifikačním prostředkem. O přirozený rod se opírá dělení na maskulina, femina a neutra (mužský, ženský

a střední rod). [5] uvádí, že gramatický rod substantiv je dán soubory skloňovacích koncovek. Zakončení substantiv na konsonant (souhlásku) je obvykle znakem maskulin (kromě typů *předseda, soudce, hajný, krejčí*). Zakončení na -a, -e/-ě je typické pro feminina kromě typů *píseň, kost, pokojská, vedoucí*. Zakončení na -o, -e/-ě je typické pro neutra kromě typů *stavení, jízdné, telecí*.

Čeština má sedm pádů (nominativ, genitiv, dativ, akuzativ, vokativ, lokál, instrumentál – tradičně se označují jako 1. – 7. pád). Jejich přehled je znázorněn v tabulce 4.2. Kromě podstatných jmen se tyto pády uplatňují i u skloňování přídavných jmen, zájmen a číslovek. Pátý pád (vokativ) se vyskytuje jenom v jednotném čísle. Pádové formy jsou základním prostředkem, který vyjadřuje funkce substantiva ve větě.

Pád		**Pádová otázka**
1. pád	nominativ	kdo? co?
2. pád	genitiv	koho? čeho?
3. pád	dativ	komu? čemu?
4. pád	akuzativ	koho? čeho?
5. pád	vokativ	oslovujeme, voláme
6. pád	lokál	(o) kom? (o) čem?
7. pád	instrumentál	kým? čím?

Tabulka 4.2: Sedm pádů v češtině.

Skloňování substantiv

Typy substantivního sklonění jsou podle [5] sdruženy do skupin podle gramatického rodu (v maskulinu dále podle životnosti) a podle povahy zakončení. Soubory koncovek tvoří paradigmata a ta jsou reprezentována vzory (viz tabulka 4.3). Pro zařazení do vzoru je důležitý rod substantiva a povaha jeho zakončení v prvním a druhém pádu jednotného čísla. V rámci jednoho vzoru bývají dále odlišeny podtypy (např. vedle vzoru *hrad podtypy* les, ostrov). Počet vzorů i jejich lexikální označní není pevně stanoveno – lze je volit různě.

Jednotlivé vzory jsou tvořeny koncovkami sedmi pádů, a to v jednotném i množném čísle. Žádný vzor však neobsahuje čtrnáct různých koncovek, existuje tu jistá homonymie. Nejvíce různých koncovek má vzor *žena*, nejméně vzor *stavení* a podtyp *paní*. U mužského rodu se tyto vzory dělí dále na životné a neživotné.

4.4.2 Slovesa (verba)

Definice v [4] uvádí, že sloveso je ohebný plnovýznamový slovní druh. Označuje dynamické (v čase probíhající) příznaky substancí. Toto vše může sloveso vyjádřit jako děj, který v okamžiku promluvy proběhl (děj minulý), nebo jako děj probíhající (přítomný) anebo jako děj, který proběhne (budoucí) [5].

Slovesa se mohou transformovat v substantiva (*kreslení, lov*), adjektiva (*běžící, odřený*) a adverbia (*letmo, mlčky*).

Tvary slovesné vyjadřující osobu, čas a způsob jsou v [5] nazývány *určitými tvary*. Ty tvary, které je nevyjadřují, nazýváme *neurčité*.

Rod	Vzor	Podtypy
mužský životný	pán muž předseda soudce	hoch, občan otec, dobyvatel paňáca, husita, sluha
mužský neživotný	hrad stroj	ostrov, zámek, domeček
ženský	žena růže píseň kost	skica, ruka ulice
střední	město moře kuře stavení	jablko, středisko, pončo bojiště

Tabulka 4.3: Přehled vzorů a jejich podtypů respektující domácí gramatickou tradici [5].

Osoba slovesná slouží k základní orientaci o účastnících promluvy vzhledem k obsahu věty. Čeština má tři gramatické osoby. První a druhá osoba vyjadřuje, že mluvčí nebo adresát jsou účastníky obsahu věty v pozici podmětu, třetí osoba znamená, že jimi nejsou.

Děj týkající se mluvčího je vyjádřen formou první osoby. Děj adresátův je vyjádřen druhou osobou. Děj neúčastněné osoby, zvířete nebo neživé substance je vyjádřen formou třetí osoby. Všechny tři gramatické osoby lze v češtině vyjádřit jak v čísle jednotném, tak množném.

Slovesným videm rozumíme ten fakt, že české sloveso existuje ve dvou podobách, které mají stejný lexikální význam, ale odlišují se od sebe vztahem k završenosti (ukončenosti) děje. Slovesa dokonavá vyjadřují, že děj buď byl ukončen, nebo že bude završen (*napsal jsem dopis, napíšu dopis*). Slovesa nedokonavá se k faktu ukončení děje nevyjadřují (*psal jsem dopis, píšu dopis, budu psát dopis*). Ne všechna slovesa mají své vidové protějšky. Slovesa modální a některá stavová jsou nedokonavá (např. *muset, chtít, vědět, smět, umět...*).

Všechny formy slovesné se tvoří různými morfy od slovesných kmenů. Tradičně se rozeznávají tři druhy kmenů [4]:

- Kmen prézentní je část slovesa ve 3. osobě singuláru po odtržení osobní koncovky: *nes-e, tiskn-e* atd.

  Podle prézentních tvarů dělíme slovesa do pěti tříd: 1. třída: *nese, bere, maže, peče, tře*; 2. třída: *tiskne, mine, tne*; 3. třída: *kryje, kupuje*; 4. třída: *prosí, trpí, sází*; 5. třída: *volá*.

- Kmen minulý je část slovesa po odtržení koncovek *-l* (*-la, -lo, -li, -ly, -la*): *nes-l, tisk(nu)-l* atd.

- Kmen infinitivní je část slovesa ve tvaru infinitivu po odtržení koncovky *-t*: *nés-t, péc-t* atd.

Adjektiva (přídavná jména)

Podle [4] jsou adjektiva plnovýznamové slovní druhy označující vlastnosti substancí (tj. osob, zvířat a neživých jevů konkrétních i abstraktních). Rod, číslo a pád adjektiv je dán shodou se substantivem, ke kterému se vztahují. Z morfologického hlediska (tj. pokud jde o druhy skloňování) se rozlišuje trojí typ adjektivní flexe: jmenná, složená a smíšená (vzory *otcův, matčin*). Složená se pak dále dělí na tvrdou (vzor *mladý*) a měkkou (vzor *jarní*).

Zájmena (pronomina)

Zájmena představují plnovýznamový ohebný slovní druh tvořený v podstatě uzavřenou množinou výrazů. Zájmena různou měrou realizují schopnost jazyka pojmenovávat substance a vlastnosti substitučně [4].

Podle [5] mohou zájmena plnit několik funkcí:

- pojmenovávají jevy ve vztahu k jejich roli v komunikaci (*já, ty, on...*)
- odkazují k referentu v rámci textu nebo ukazují mimo text (*on, ona, ten, tento...*)
- vyjadřují vztah komunikační role a posesivity (*můj, tvůj, svůj...*)
- vyžadují doplnění informací o předmětu komunikace (*kdo, co, jaký, který, čí*)
- vyjadřují vztah (*kdo, co, jaký, který, čí, jenž*)
- vyjadřují různé odstíny neurčitosti předmětu, vlastnosti nebo vlastníka (*někdo, ledajaký, čísi...*)
- popírají předmět nebo vlastníka (*nikdo, nic, žádný, ničí*)
- identifikují (*týž, jiný*)
- totalizují a kvantifikují (*všichni, každý*)
- reflexivizují *se, svůj*

4.4.3 Další slovní druhy

Číslovky (numeralia)

V [5] je uvedeno, že číslovky slouží jako slovní druh vyjadřující kvantovost, a to buď počítanou (určitou, vyjádřitelnou čísly), nebo nepočítanou (neurčitou, čísly nevyjádřitelnou).

Kvantovost nepočítanou vyjadřují číslovky *mnoho, málo, několik* a hovorově i jiná slova v jejich funkci (*moře, fůra, trocha*).

Nemají přesně stanovené a stanovitelné hranice, neboť kvantitu mohou vyjadřovat i substantiva (*desetina, stovka*), adjektiva (*poslední*), příslovce (*moc, trochu*) a zájmena (*tolik, kolik*) [4].

Specifickou slovotvornou vlastností číslovek je, že většina vyšších číselných hodnot se označuje slovními spojeními (kombinované číslovkové výrazy – *tisíc devět set devadesát devět*) [4].

Příslovce (adverbia)

Definice v [4] uvádí, že tento neohebný plnovýznamový slovní druh vyjadřuje především okolnosti, za kterých se realizuje obsah slovesa, za nichž platí obsah adjektiva anebo dalšího adverbia: *pracuje rychle, přijel večer, raně gotický, stále unavený, velmi mladě* atd.

Podle druhu okolnostního významu, který vyjadřují, dělíme adverbia na příslovce místa, času, způsobu a příčiny.

Předložky (prepozice)

Podle [4] je tento slovní druh neohebný a významově nesamostatný. Předložky jsou součástí pádových podob substantiv (a dalších slovních druhů), čímž spoluvytvářejí významy jmen. Důležitou roli u předložek hraje jejich spojitelnost s různými pády substantiv.

Soustava jednoslovných předložek je dnes prakticky uzavřena. V jazyce však existuje trvalá potřeba vyjadřovat měnící se a zpřesňující se povahu vztahu, a proto vznikají nové jednoslovné i víceslovné předložkové výrazy.

Spojky (konjunkce)

Spojky jsou neohebný neplnovýznamový slovní druh tvořený v podstatě uzavřenou množinou výrazů. Slouží ke spojování celých vět nebo i jen částí vět. Spojky tedy vyjadřují jejich vzájemný vztah [4].

Spojky se mohou vyskytovat v několika formách: jednoslovná (*a, že, když*), opakovaná (*a, a, a*), dvojitá (*buď – nebo*), nebo víceslovná (vstupují do ní další slovní druhy).

Potřeba vyjadřovat nové vztahy, nebo staré upřesňovat vede ke vzniku kombinovaných spojkových výrazů: *hned (když), jenom (aby), (až) konečně* apod.

Částice

Částicemi se v české lingvistické tradici rozumí velmi různorodá skupina neohebných neplnovýznamových výrazů, která nezahrnuje výrazy spojkové a předložkové. S ohledem na svou neplnovýznamovost však bývají častice někdy řazeny do jedné skupiny spolu se spojkami a předložkami. Zástupci této skupiny neohebných výrazů pak bývají nerozlišně označovány jako *partikule* [5].

Citoslovce (interjekce)

Podle [5] představují citoslovce neohebný slovní druh se zvláštními typy významů. V komunikaci jsou plně sdělná a formálně samostatná. Považují se za jeden z faktorů výstavby textu, a to svými funkcemi emociálními a sociálními (kontaktovými).

Citoslovce vyjadřují bezprostřední reakci na podněty. Některé z těchto podnětů budí různé city, pocity, hodnocení a jejich výrazem jsou subjektivní citoslovce pocitová (*hurá, tralala, au(vajs), hergot, kčertu...*).

Citoslovce zvukomalebná jsou reakcí na nějaký zvuk, a to snahou o napodobení tohoto zvuku(*vrz, břink, prásk, haf-haf...*).

Třetí skupinu tvoří obecně kontaktová a vybízecí citoslovce (*hej, haló, panečku, bacha, tumáš, marš, že ano...*).

4.5 Slovotvorba

Dle definice v [4] je slovotvorba gramatická disciplína, zabývající se jednak procesem tvoření slov na základě slov již existujících z hlediska způsobů a postupů, jednak formou, významem a fungováním výsledků těchto procesů.

V [5] je uvedeno, že nově vytvořená slova bývají členitá, protože se skládají jednak ze základu už existujícího pojmenování, které tu plní funkci slova motivujicího, jednak ze složek, které na tomto základě konstituují význam nový. Při jeho vytváření mohou mít tyto složky povahu morfémů, a pak se jedná o derivaci (odvozování), nebo se může autosémantický slovní základ kombinovat s jiným a jde o kompozici (skládání slov).

U derivátů může odvozovací morfém stát za základem, a pak má název suffix (přípona), např. *umýva-dlo*. Tento způsob tvoření nazýváme sufixací. Součástí sufixu bývá u flektivních slovních druhů zpravidla i koncovka, která určuje slovní druh vzniklého slova, např. *umýva-dl-o* = substantivum, neutrum, vzor *město* [5].

Resufixace je náhrada jednoho sufixu druhým, který má jinou funkci, např. *vlas-atý* → *vlas-áč*.

Stojí-li derivační morf před základem, jde o prefix (předponu), např. *při-volat*, a proces jeho připojení se nazývá prefixace. Sufixy i prefixy mají souhrnný název afixy.

Postfix je název pro takový sufix, který se připojuje až na sám konec slova za jeho koncovku a tam zůstává i při flexi, např. *jaký-si, jakého-si...*

Podle [4] je kompozice v češtině méně významná než derivace.Při tvoření slov kompozicí se v jednom slově spojují dva nebo více slovních základů, které bývají spojeny pomocí tzv. konektu, např *život-o-pis*, který ovšem může být i nulový, např. *knih-tisk*. Méně často má přední člen podobu flektivního tvaru, např. *lásky-plný*. Často dochází k tomu, že je kompoziční postup kombinován s derivačním [5]. Spojení dvou slovních základů tedy bývá zároveň opatřováno sufixem, např. *dřev-o-rub-ec*. Za kompozita už nepovažujeme odvozeniny z kompozit, např. *dřevorubec-ký*.

Při uvedených slovotvorných postupech dochází často k tomu, že se hláskový sklad slovních základů obměňuje, např. *plech* → *plíš-ek, zasypat* → *zásyp*. Tento jev se jmenuje hlásková alternace. Alternace se mohou týkat jak konsonantů, tak vokálů.

Dále se v [5] uvádí, že alternace konsonantů bývají pravidelnější, např. při připojení sufixu začínajícího hláskou *i* alternují vždy *d : ď, g : ž, h : ž, ch : š, k : č, n : ň, r : ř, t : ť*. Tyto alternace označujeme jako *alternace typu i*.

Alternace vokálů se týkají jednak jejich kvantity, např. *a : á, e : é, i : í, y : ý, u : ú*, jednak kvality, např. *o : ů, u : ou, ě/e : í e : í/ý* apod. Tyto alternace nejsou vždy pravidelné.

Zvláštní případ představují tzv. vznikové a zánikové alternace, prakticky střídání *e : 0 (pes* → *psí)* a *0 : e, 0 : é (teplo* → *tepel-ný, jádro* → *jadér-ko)*. K vznikové alternaci dochází hlavně tam, kde by v důsledku připojení sufixu vznikla skupina tří (nebo více) konsonantů.

Slovotvorným základem může být pouhý kořen slova (*plav-ba*), nebo kořen s prefixem (*roz/plav-ba*), nebo kořen s kmenotvornou příponou (*plov/a-cí*), nebo kořen se sufixem (*plav/ec-ký*), kompozitum se sufixem (*par/o/plav/eb-ní*) atd. Slovotvorným základem tedy rozumíme jakkoli komplikovanou (nebo naopak jednoduchou) strukturu, která slouží jako motivující formace k vytvoření nového slova.

V jakkoli dlouhém slovotvorném řetězci je základem tvroby dalšího derivátu vždy až jeho předposlední stupeň, tedy např. *umět* → *rozumět* → *srozumět* → *srozumitelný* → *srozumitelnost*. Slovotvorným základem slova *srozumitelnost* je tedy adjektivum srozumitelný.

U sloves je situace odlišná, protože repertoár slovesných sufixů je velmi úzký. Jako odvozovací sufixy slouží ve většině případů pouze infinitivní kmenotvorné přípony a sufix

-va-, zřídka také sufix *-k-*. Proto musejí být tyto afixy až na výjimky polyfunkční. Z toho tedy vyplývá skutečnost, že u sloves hraje prefixace podstatně významnější roli než u jiných slovních druhů.

Vzhledem k mechanismu historického slovotvorného rozvoje obsahuje současný stav slovotvorby více nepravidelností než plány ostatní. Většina slov jednou utvořených, pokud jich bylo zapotřebí, žila v jazyce i poté, co se způsob, jímž vznikla, stal neproduktivním.

Generování slov

Deriváty slov odvozené z určitého kmene lze tvořit také algoritmicky. V [13] je uvedeno, že při formálním popisu derivace slov však nelze jednoznačně rozhodnout, jestli je daný sufix možné kombinovat s daným kmenem. Dochází k tzv. nadgenerování slov (*overgeneration*). Některá z takto vzniklých slov se však v jazyce mohou alespoň ojediněle vyskytovat, jejich užití závisí na individuálním rozhodnutí uživatele.

Algoritmicky vytvářená odvozená slova můžeme podle [13] členit podle stupnice: slova uvedená v některém výkladovém slovníku češtiny nebo doložená v korpusech, slova nedoložená, ale vzhledem k jejich slovotvorné stavbě použitelná, slova kuriózní, slova nepoužitelná.

Kapitola 5

Snowball

Snowball [9] je malý jazyk pro práci s řetězci. Jeho jméno bylo zvoleno jako pocta jazyku SNOBOL. S tímto jazykem má také společnou základní myšlenku, a to že určité vzory řetězců vysílají signály, které jsou využívány k řízení běhu programu.

Snowball je jazyk sloužící ke snadnému a přesnému popisu lemmatizačních algoritmů. Kompilátor jazyka Snowball překládá skripty na programy v jazyce Java nebo ANSI C. K tomuto jazyku existuje manuál a mnoho skriptů lemmatizačních algoritmů, které slouží jako ukázkové příklady. Jedná se o jednoduchý jazyk, kterému se zkušený programátor naučí rychle.

Jedním z důvodů pro vznik jazyka Snowball bylo, že neexistovaly snadno dostupné lemmatizační algoritmy pro jiné jazyky než angličtinu. Usilovně se pracovalo na vývoji a vyhodnocování těchto algoritmů, ale přesto nebyla k dispozici žádná jejich jednoznačná algoritmizovaná forma, ze které by se dala snadno vytvořit implementace v jazyce C, Java, Perl atd.

5.1 Kompilace a spuštění

Program Snowball je možné stáhnout z internetových stránek projektu [9]. Zdrojové soubory obsahují i `makefile`, takže kompilace proběhne standardně po zadání příkazu `make`. Chování programu Snowball lze ovlivnit několika parametry. K nejdůležitějším patří parametr `-java`. Pokud je při spuštění programu zadán, bude Snowball generovat zdrojové kódy pro jazyk Java. Pokud zadán není, budou se generovat zdrojové kódy pro jazyk C. Vyčerpávající popis dalších přepínačů včetně příkladů jejich použití je uveden na internetových stránkách projektu [9].

Distribuce obsahuje také zdrojové kódy ukázkového programu `stemwords`. Jedná se o jednoduchou konzolovou aplikaci umožňující lemmatizaci slov načítaných ze souboru nebo ze standardního vstupu. Parametrem `-h` je možné si nechat zobrazit nápovědu k ovládání aplikace. Prostudováním zdrojového souboru aplikace `stemwords` je možné se seznámit s používáním funkcí generovaných programem Snowball. Celé API (*Application Programming Interface*) jazyka Snowball je popsáno na internetových stránkách projektu [9].

Skripty v jazyce Snowball je možné také debuggovat. Jednou z možností je použití parametru `-syntax` při spuštění programu Snowball. Program poté bude na výstup přehledně vypisovat, jaké funkce jsou v jaký moment volány. Další možností je použít přímo ve skriptu příkaz `?`, což bude mít za následek volání funkce `debug`(...) Tato funkce je definována v souboru `runtime/utilities.c` (zpravidla je však zakomentovaná). Ve slovech posílaných na

výstup budou označeny všechny důležité body (aktuální poloha kurzoru, limitní pozice atd.). V současnosti debugger neposkytuje žádnou možnost, jak hlásit hodnotu proměnných typu integer nebo boolean. Jediným způsobem jak toho dosáhnout, je upravit ručně zdrojové soubory a na patřičná místa vložit kontrolní výpisy.

5.2 Znakové sady

Ve skriptech v jazyce Snowball je nutné explicitně deklarovat kódy všech využívaných non-ASCII znaků. Například ve zdrojovém kódu českého stemmeru musí být zahrnuty tyto řádky:

```
/* special characters (in UTF-8) */
stringdef a'   hex '0e1'
stringdef cv   hex '10d'
stringdef dv   hex '10f'
stringdef e'   hex '0e9'
stringdef ev   hex '11b'
stringdef i'   hex '0ed'
stringdef nv   hex '148'
stringdef o'   hex '0f3'
stringdef rv   hex '159'
stringdef sv   hex '161'
stringdef tv   hex '165'
stringdef u'   hex '0fa'
stringdef u^   hex '16f'
stringdef y'   hex '0fd'
stringdef zv   hex '17e'
```

V kódování UTF-8 odpovídají hexadecimální hodnoty `0e1, 10d, 10f` znakům *á, č, ď* atd. Toto kódování ve skriptech poté koresponduje s kódováním použitým pro vstupní data výsledné aplikace. Výše uvedený příklad odpovídá konvencím uvedeným v dokumentaci jazyka Snowball. Čárka nad písmenem je reprezentována *apostrofem*, háček písmenem *v* a kroužek znakem ^.

5.3 Základní principy jazyka

Informace v této kapitole pochází z manuálu jazyka Snowball dostupného na internetových stránkách projektu [9].

Základní myšlenkou všech skriptů v jazyce Snowball je aplikování určité funkce na řetězec. Řetězce mohou být zpracovány jak zleva doprava, tak i zprava doleva. Každý řetězec je popsán dvěma význačnými body. Prvním je *kurzor* označovaný písmenem *c*, druhým hraniční bod (limit) značený písmenem *l*. Během zpracování řetězce se kurzor posunuje směrem k limitu. Pohyb kurzoru může být ovlivněn konstrukcemi and, or, not atd. Ty mohou způsobit i to, že se kurzor bude pohybovat zpět. Při inicializaci je kurzor umístěn na začátek řetězce a limit na konec.

```
' t | r | a | k | t | o | r | i | s | t | a '
0   1   2   3   4   5   6   7   8   9  10  11
c                                           l
```

Kurzor i limit ukazují vždy na hranice mezi znaky a ne přímo na znaky samotné. Všechny znaky mezi kurzorem a limitem jsou označovány jako *c:l*. Na kurzor a limit je možné pohlížet jako na numerické hodnoty číslované od nuly. Díky tomu se jejich hodnota může nastavovat v aritmetických výrazech.

Ve výše uvedených příkladech byl vždy kurzor umístěn nalevo od limitu a pohyboval se doprava. Tento proces může probíhat i druhým směrem. Při zpětném zpracování se vymění pozice kurzoru a limitu a kurzor se pohybuje doleva směrem k limitu.

Snowball umožňuje využívat ve skriptech procedury. Nová procedura se definuje zápisem `define R as C`, kde `R` znamená název procedury a `C` v závorkách uzavřený seznam nula a více příkazů. Snowball v současné době nepodporuje procedury se vstupními parametry.

Řízení běhu programu ve Snowballu není zpracováno konstrukcemi *if, then, break* apod. jako v jazyce C, ale využitím signálů. Každý vykonaný příkaz vrací hodnotu *true* nebo *false* a na základě těchto návratových hodnot je poté řízen běh progamu. Mějme např. tyto dva příkazy: `$x = 1` a `$x > 0`. První příkaz nastaví proměnnou `$x` na hodnotu 1 a bude vždy vracet signál true. Druhý příkaz testuje, jestli je hodnota proměnné `$x` větší než 0. Pokud ano, vrací signál true, jinak vrací signál false.

Sekvence několika příkazů se může chovat jako příkaz jediný, pokud bude celá sekvence uzavřena do závorek. Mějme např. následující posloupnost příkazů: $(C_1C_2C_3 \dots C_iC_{i+1})$. V tomto případě bude příkaz C_{i+1} zpracován, pouze pokud každý z předcházejících příkazů $C_1 \dots C_i$ vrátí signál true. Ale jakmile některý z příkazů C_i vrátí signál false, budou následující příkazy $C_{i+1}C_{i+2} \dots$ ignorovány a celá sekvence bude také vracet signál false. Pokud vrátí každý z příkazů C_i signál true, pak samozřejmě i celá sekvence vrátí signál true.

Příkazy je možné kombinovat pomocí následujících konstrukcí:

C_1 or C_2	Vykoná C_1. Pokud vrátí true, ignoruje C_2, jinak vykoná C_2. Výsledný signál je true právě když C_1 nebo C_2 vrátí true.
C_1 and C_2	Vykoná C_1. Pokud vrátí false, ignoruje C_2, jinak vykoná C_2. Výsledný signál je true právě když C_1 a C_2 vrátí true.
not C	Vykoná C. Výsledný signál je true, když C je true. Jinak vrací false.
try C	Vykoná C. Výsledný signál bude vždy true. Nezáleží na signálu C.
fail C	Vykoná C. Výsledný signál bude vždy false. Nezáleží na signálu C.

5.4 Příkazy pro práci s řetězi

Pokud je v proměnné `$s` uložen řetezec, pak zápis `$s C` způsobí aplikování příkazu `C` na daný řetězec. Pokud příkaz `C` vrátí true, nastaví se kurzor na novou pozici. Pokud vrátí false, nebude pozice kurzoru definována.

Základní testy

Jazyk Snowball umožňuje provádět následující operace s řetězci:

- S – `S` znamená buď řetězec nebo proměnnou s řetězcem. Pokud úsek `c:l` začíná podřetězcem `S`, nastaví se kurzor na konec tohoto podřetězce a vrací se signál true. Jinak se vrací false.

- C_1 or C_2 – Funguje stejně jako příklad pro čísla uvedený výše, ale navíc pokud C_1 vrátí false, bude před provedením C_2 kurzor nastaven zpět na jeho původní pozici před vykonáním C_1. Díky tomu se začne příkaz C_2 vykonávat ze stejného místa jako příkaz C_1.
- C_1 and C_2 – Poté co C_1 vrátí true, je kurzor obdobně nastaven zpět na jeho původní pozici a poté je vykonán C_2.
- test C – Vykoná se příkaz `C`, ale nebude změněna poloha kurzoru. Vrací stejný signál jako příkaz `C`.
- do C – Vykoná příkaz `C`, nastaví kurzor zpět na jeho původní pozici a vrátí true.
- goto C – Kurzor je posunován doprava, dokud `C` vrací true. Pokud už kurzor nemůže být dále posouván vpravo, protože narazil na limit, bude vrácen signál false. Kurzor se vždy vrátí na pozici před vzor, který vyhověl porovnání.

  ```
  x goto 'tor'          /* kurzor bude na pozici za 'trak' */
  ```

- gopast C – Podobné jako `goto`, ale kurzor se nevrací zpět, takže zůstává za vzorem, který vyhověl porovnání.

  ```
  x gopast 'tor'        /* kurzor bude na pozici za 'traktor' */
  ```

- repeat C – Opakuje příkaz `C`, dokud nevrátí false. Až tento okamžik nastane, vrátí kurzor zpět na pozici, kterou měl před posledním opakováním `C`. Poté celá funkce vrátí true.

  ```
  x repeat gopast 'a'     /* kurzor bude na pozici za posledním 'a' */
  ```

- loop AE C – Odpovídá sekvenci `C C` ... `C` zapsané AE-krát, kde AE je aritmetický výraz. Tato konstrukce má stejný význam jako cyklus for v jazyce C.
- atleast AE C – Tento zápis je ekvivalentí konstrukci `loop AE C repeat C`.
- hop AE – Posune pozici kurzoru o AE znaků směrem k limitu. Pokud je AE záporné, nebo je v úseku `c:l` méně než AE znaků, vrací false. Např. příkaz `test hop 3` otestuje, jestli úsek `c:l` je delší než dva znaky.
- next – Tento zápis je ekvivalentní konstrukci `hop 1`.

Operace s textem

S řetězci se dá pracovat precizněji, než pouze vkládat je celé do proměnné, jak bylo ukázáno výše. Základem je definování podřetězce – tzv. *slice*.

- `[` – Nastaví levý okraj podřetězce slice na aktuální polohu kurzoru.
- `]` – Nastaví pravý okraj podřetězce slice na aktuální polohu kurzoru.
- `->` s – Zkopíruje obsah podřetězce slice do proměnné `s`.

- `<-` S – Nahradí obsah podřetězce slice proměnnou (nebo řetězcem) `S`.

```
/* předpokládejme, že proměnná $x obsahuje 'traktorista'
$x (  [                    // '[traktorista'
      loop 2 gopast 'r'    // '[traktor|ista' - kurzor označen '|'
      ]                    // '[traktor]ista'
      -> y                 // y obsahuje 'traktor'
   )
```

- delete – Smaže aktuální podřetězec slice. Ekvivalentní zápisu `<- ''`.
- insert S – Vloží `S` před kurzor. Posune kurzor vpravo od vloženého `S`.
- attach S – Stejné jako `insert`, ale po vložení nepřesune kurzor.

Značky v textu

Pro práci s řetězci je v jazyce Snowball možné využít systém značek.

- setmark X – Vloží do proměnné `X` aktuální číselnou hodnotu kurzoru.
- tomark AE – Posune kurzor na pozici určenou aritmetickým výrazem AE.
- atmark AE – Testuje, jestli je kurzor na pozici AE. Vrací true nebo false.
- tolimit – Posune kurzor na pozici limitu. Vždy vrací true.
- atlimit – Testuje, jestli je kurzor na pozici limitu. Vrací true nebo false.

Zpětné zpracování

Všechny výše uvedené příkazy počítaly s umístěním kurzoru nalevo od limitu a posunováním kurzoru směrem doprava. Tento postup se však může také obrátit.

Pokud má procedura fungovat ve zpětném režimu, musí byt definována uvnitř bloku `backwardmode{...}`.

- backwards C – Dochází k záměně kurzoru s limitem. Kurzor se tedy pohybuje doleva směrem k limitu. Vrací se signál příkazu `C`. Po vykonání příkazu se kurzor vrací na svou původní pozici.
- reverse C – Kurzor se nastaví na pozici limitu a poté se místo doprava pohybuje doleva k počátku řetězce. Příkaz `C` nesmí obsahovat mazání nebo vkládání znaků.

Detekování podřetězců

Konstrukce `substring` a `among` je základním stavebním kamenem stemmovacích algoritmů. Funguje obdobně jako konstrukce `switch case` v jazyce C. V nejjednodušší formě může být zapsána takto: `substring among('S1' 'S2' 'S3' ...)`. V tomto případě začne od pozice kurzoru vyhledávat nejdelší shodný podřetězec z uvedených `'S1'`, `'S2'` nebo `'S3'` (všechny řetězce S_i musí být navzájem různé).

Kapitola 6

Požadavky na aplikaci

Cílem práce bylo v jazyce Snowball implementovat stemmer českého jazyka založený na gramatických pravidlech. V současné době existují stemmery v jazyce Snowball pro 16 různých světových jazyků [9], ale čeština mezi nimi není. Stemmer nemusí generovat přesný morfologický kořen slova, ale měl by všechny možné tvary jednoho slova převádět na stejný tvar (tzv. stem). Pro češtinu existují pouze dva brute force stemmery pracující se slovníkem, ve kterém mají ke každému slovu přiřazen jeho základní tvar.

V článku [11] je uvedeno, že tohoto principu využívá např. fulltextové vyhledávání v databázi PostgreSQL. Tato technologie je založena na projektu *Tsearch2* [2], což je fulltextový systém ukládající metainformace o dokumentech v klasické relační databázi. Stemming je zde prováděn na základě dat z českého slovníku Ispellu[1]. Problémem uvedeného řešení je velikost slovníku, resp. její vliv na rychlost vyhledávání. Situaci, kdy hledané slovo není ve slovníku uvedeno, řeší autoři tak, že vrací zadané slovo v nezměněné podobě, pouze ho převedou na malá písmena.

Zajímavý přístup pro stemmování češtiny zvolili indičtí autoři v [8]. Jejich statistický stemmer YASS nepoužívá žádná gramatická pravidla, pouze pomocí vzdálenostní funkce počítá, jak moc jsou si dva řetězce podobné. Vzdálenostní funkce pracuje se dvěma řetězci „s“ a „t“ a s reálným číslem „r“. Čím menší je hodnota „r“, tím víc jsou si řetězce „s“ a „t“ podobnější. Protože se jedná o statistický stemmer, musí před jeho spuštěním na novém jazyce proběhnout trénovací fáze. Výsledky tohoto stemmeru pro několik různých jazyků jsou srovnatelné s výsledky stemmerů založených na gramatických pravidlech. Autoři sice neměli pro češtinu k dispozici dostatek trénovacích dat, i přesto však jejich stemmer poskytuje solidní výsledky.

Otevřené slovníky české gramatiky

Hlavní částí českých dat pro Ispell je soubor zhruba 100 000 sloves, příslovcí odvozených od přídavných jmen a pravidelně skloňovaných podstatných a přídavných jmen. Dále obsahuje seznam 150 nepravidelných a 350 nesklonných podstatných a přídavných jmen, z nichž některá by mohlo být vhodné začlenit do stemmeru češtiny.

Dalšími součástmi jsou seznamy různých vlastních jmen (jména, příjmení, jména národů či obcí...), cizích jmen, případně ostatních názvů. Tato část nebude z hlediska tvorby stemmeru zajímavá, protože takováto slova zpravidla nevznikají odvozováním na základě gramatických pravidel.

[1] Český slovník pro Ispell je dostupný na adrese `ftp://ftp.tul.cz/pub/unix/ispell/`

Poslední částí českých dat pro Ispell jsou soubory zájmen, spojek, citoslovcí, částic a různých zkratek. Tyto informace sice nenajdou uplatnění při tvorbě stemmeru, ale na jejich základě jsem sestavil seznam stopslov pro češtinu. Seznam jsem následně dále doplňoval na základě informací získaných v knize [5]. Celkem seznam obsahuje téměř 1 200 položek.

Na základě dat pro český Ispell je vytvořen také slovník pro kontrolu pravopisu v OpenOffice[2]. Hlavní součástí je seznam více než 170 000 českých podstatných a přídavných jmen, sloves a příslovcí. Dále obsahuje seznam zhruba tisíce sufixů včetně regulárními výrazy zapsaných pravidel pro jejich používání ve slovech.

Příprava testovacích dat

Abych mohl výsledky stemmeru nějakým způsobem testovat, sestavil jsem pro každý slovní druh testovací sadu dat. Při sestavování sady pro podstatná jména jsem se snažil pokrýt maximální část gramatických pravidel, na jejichž základě jsou slova vytvářena.

Informace jsem čerpal z [5]. Pro každou kategorii sufixů jsem vybral několik zástupců dané kategorie. Tak vznikl seznam 277 podstatných jmen v prvním pádě jednotného čísla. Poté bylo nutné získat také tvary slov v ostatních pádech. Proto jsem ke každému slovu doplnil, jakého vzoru (případně podtypu) je. Všech 48 vzorů a podtypů včetně všech jejich vyskloňovaných variant použitých pro vývoj stemmeru se nachází na přiloženém CD v adresáři `scripts/endings/nouns`. Na základě znalosti jakého vzoru dané slovo je a jaké jsou koncovky tohoto vzoru ve všech pádech, byly ke každému slovu vygenerovány všechny jeho existující tvary. Celkem tedy testovací sada podstatných jmen obsahovala 2 314 slov.

Následně byla ke každému slovu doplněna správná podoba výsledného stemu. Stemmer však nemusí vracet pouze přesnou podobu kořene slova, proto je u každého slova uveden i částečně správný výsledek, který by také mohl být správným stemem.

Při sestavování testovacích sad pro přídavná jména a slovesa jsem postupoval obdobným způsobem. U přídavných jmen bylo pro získání koncovek jednotlivých pádu využito 14 vzorů, u sloves 18. Jejich přehled se opět nachází na přiloženém CD.

Protože čeština je jednak jazyk složitý a jednak zdaleka ne všechna česká slova vznikla odvozením ze slov jiných, rozhodl jsem se sestavit ještě druhou testovací sadu dat, která by se nespoléhala jen na generování slov na základě gramatických pravidel, ale odrážela by reálně používanou češtinu. Ze tří náhodně vybraných novinových článků o celkové délce přes 1 000 slov jsem vybral všechna podstatná jména, přídavná jména a slovesa. Po odstranění ostatních slovních druhů zůstalo 574 slov. Obě testovací sady se nachází na přiloženém CD.

Vyhodnocování správnosti výsledků

O počítání úspěšnosti výstupů stemmeru se stará skript `results.pl`, který je uložen na přiloženém CD v adresáři `scripts`. Aby mohla být analýza výsledků provedena, musí soubor se vstupními daty obsahovat také správné a částečně správné tvary stemů, a to v podobě *hledané slovo/správný výsledek/častečně správný výsledek*, kdy se na každém řádku nachází jedno slovo. Např.:

```
traktorista/traktor/trakt
traktoristův/traktor/trakt
...
```

[2] OpenOffice slovníky jsou dostupné na adrese `http://lingucomponent.openoffice.org/`

Další podmínkou je spuštění demonstračního programu s parametrem `-p2`, který pak bude výstup formátovat tak, že na každém řádku bude uvedeno jedno zdrojové slovo oddělené mezerami od výsledného stemu:

```
traktorista     traktor
traktoristův    traktor
...
```

Vyhodnocovací skript se spouští příkazem:

```
./results.pl <zdrojový soubor> <soubor s výsledky> [print]
```

Parametr `print` je nepovinný. Při jeho zadání bude skript na výstup vypisovat všechny chybné výsledky.

Výstup skriptu je ukázán na následujícím příkladě:

```
správně:    7      (70.0 %)
špatně:     3 z 10 (30.0 %)
            under-stemming: 2
            over-stemming:  1

traktoristou                    traktor
traktorista                     traktor
traktoristech                   trak
traktoristy                     traktor
traktoristu                     traktor
traktoristovi                   traktoristov
traktoristům                    traktoristum
traktoristo                     traktor
traktoristů                     traktor
traktoristé                     traktor
```

Správným kořenem všech v příkladu uvedených slov má být „traktor“. Z celkového počtu 10 slov bylo tedy 7 výsledků správných a 3 výsledky chybné. Slovu „traktoristech“ byl přiřazen kořen „trak“. V tomto případě se jedná o *over-stemming*, protože ze slova bylo odebráno příliš mnoho znaků. U slov „traktoristovi“ a „traktoristům“ bylo naopak odebráno málo znaků a jedná se tedy o dva případy *under-stemmingu*.

Při sestavování testovacích sad dat bylo vždy nutné uvést, jak mají vypadat správné a také případné částečně správné výsledky. Cílem projektu však nebylo hledat přesné kořeny slov, ale stemy (nejdelší možné společné části slov). Proto je u každého slova uveden i částečně správný výsledek. A právě jeho určení je značně individuální a problematické.

Z tohoto důvodu jsem se rozhodl pro zavedení dalšího způsobu testování stemmeru. Místo kontrolování, jestli je výsledek správný nebo chybný, budu počítat, na kolik různých stemů byly převedeny všechny tvary jednoho slova. Mějme např. tento příklad:

```
strojírně    -> strojir
strojírno    -> strojir
strojírnám   -> strojir
```

Stemmer všem třem variantám slova *strojírna* přiřadil stem *strojir*, což je ale započítáno jako 3 špatné (eventuálně částečně správné) výsledky, protože správným kořenem má být

stroj. Přesto byly všechny varianty slova *strojírna* převedeny na stejný stem, takže výsledky by vlastně měly být zcela správné.

O počítání množství stemů, na něž jsou převedeny všechny varianty jednoho slova, se stará skript `results2.pl`, který je uložen na přiloženém CD v adresáři `scripts`. Toto testování má smysl pouze na první (uměle vytvořené) sadě dat, protože ta, na rozdíl od testovací sady založené na novinových článcích, obsahuje pro každé slovo všechny jeho existující varianty.

Tento vyhodnocovací skript se spouští příkazem:

```
./results.pl <soubor s výsledky>
```

Výstupem skriptu může být např. následující tabulka:

```
1   70%
2   22%
3    8%
```

Uvedené údaje znamenají, že 70 % z celkového počtu slov bylo převedeno na jediný stem (což je očekávané chování stemmeru – ze všech variant jednoho slova vznikne stejný stem). 22 % slov bylo převedeno na dva různé stemy a 8 % slov na tři různé stemy.

Kapitola 7

Implementace

Snowball umožňuje překládat skripty buď do jazyka Java, nebo do jazyka C. Já jsem se rozhodl implementovat algoritmus v jazyce C. Stemmer bude pracovat s texty v kódování UTF-8.

Adresářová struktura, kterou v projektu používám, vychází z doporučení v distribuci programu Snowball.

- Adresář `examples` obsahuje zdrojové kódy demonstračního programu umožňujícího spouštět stemmery vytvořené v jazyce Snowball.
- Adresář `libstemmer` slouží jednak k uložení zdrojových souborů zpřístupňujících základní stemmovací funkcionalitu (*libstemmer.c*), jednak pro definici jazyků, s kterými bude možné pracovat (*modules.h*).
- V adresáři `runtime` se nachází API jazyka Snowball. Mimo jiné je zde i soubor utilities.c, ve kterém je nutné odkomentovat funkci `debug()`, pokud chceme ve Snowball skriptech využívat debuggování.
- V adresáři `snowball` je umístěn zkompilovaný program Snowball (kompilace je popsána v kapitole 5.1) a dalé stemmery napsané v jazyce Snowball (soubory czech.sbl, english.sbl a german.sbl)
- Do adresáře `src_c` ukládá program Snowball skripty přeložené do jazyka C.
- Soubor `Makefile` umožňuje snadné vytvoření demonstračního programu.

7.1 Vytvoření modulu pro češtinu

Při vytváření nového stemmeru jsem musel nejdříve realizovat několik přípravných kroků. Po stažení Snowballu a rozdělení souborů do výše uvedené adresářové struktury bylo nutné opravit ve zdrojových souborech cesty ke vkládaným hlavičkovým souborům.

Dále jsem do souboru `libstemmer/modules.txt` přidal informaci o tom, že budu pro češtinu vytvářet nový stemmer pracující s kódováním UTF-8:

```
czech     UTF_8     czech,cz
```

Adresář `libstemmer` obsahuje mimo jiné také skript `mkmodules.pl`, jehož úkolem je na základě údajů uvedených v souboru `modules.txt` vygenerovat příslušný hlavičkový soubor `modules.h`.

Skript `mkmodules.pl` se spouští takto:

```
./mkmodules.pl  modules.h
                ../src_c
                modules.txt
                libstemmer_c.in
```

Bohužel ve výsledném souboru `modules.h` vygeneruje nesprávně cesty k includovaným souborům, takže se musí ručně upravit do této podoby:

```
#include "../src_c/stem_UTF_8_czech.h"
#include "../src_c/stem_UTF_8_english.h"
```

Následně jsem upravil soubor `Makefile`. Bylo nutné přidat cestu k Snowballem generovaným zdrojovým kódům mého stemmeru (uloženy v adresáři `src_c`) a opravit cesty k některým dalším includovaným souborům.

Také jsem procházel zrojový kód demonstračního programu `examples/stemwords.c` dodávaného přímo se Snowballem a zjistil jsem, že pro mé potřeby bude jeho funkcionalita dostačovat. Našel jsem v něm pouze jeden problém – neumí převádět velká písmena s diakritikou na malá. V programu jsem nakonec provedl jen dvě drobné úpravy. Jako výchozí jsem nastavil český stemmer s kódováním UTF-8 a přidal jsem možnost mít ve vstupních datech komentáře. Veškerý text od znaku `/` nebo `#` až do konce řádku je chápán jako komentář a nebude stemmerem zpracováván.

Zdrojový kód stemmeru v jazyce Snowball ukládám do souboru `snowball/czech.sbl`. Snowball z něj tímto příkazem:

```
./snowball  czech.sbl
            -o ../src_c/stem_UTF_8_czech
            -ep czech_UTF_8_
            -r ../runtime/
            -u
```

vygeneruje do adresáře `src_c` zdrojový soubor `stem_UTF_8_czech.c` a hlavičkový soubor `stem_UTF_8_czech.h` (parametr `-o`). Jednotlivé funkce stemmeru budou mít ve svém názvu prefix `czech_UTF_8_` (parametr `-ep`). Další parametry udávají cestu k API Snowballu (`-r`) a vynucují si použití kódování UTF-8 (`-u`).

Pro kompilaci demonstračního programu je připraven soubor `Makefile`. Návod k ovládání programu se zobrazí po spuštění s parametrem `-h`. Program může načítat slova buď ze standardního vstupu nebo ze souboru.

Postup při tvorbě stemmeru

Na internetových stránkách Snowballu je k dispozici 19 stemmovacích algoritmů pro různé jazyky. Každý z nich obsahuje zdrojové soubory v jazyce Snowball, stručné vysvětlení činnosti algoritmu, ukázku slovní zásoby v daném jazyce a výsledky stemmování těchto slov.

Tyto algoritmy mohou kromě oficiálního manuálu jazyka Snowball sloužit jako další výukový materiál. Nejpodrobněji je popsán stemmer pro angličtinu, jehož autorem je Martin

Porter – tvůrce Snowballu. Porter nejdříve popisuje, jak funguje jeho stemmovací algoritmus a následně detailně vysvětluje, jak jednotlivé kroky zapsat v jazyce Snowball. Před čtením tohoto návodu je však vhodné mít nastudované informace z manuálu Snowballu, jinak bude význam některých pokročilých konstrukcí hůře pochopitelný. V jazyce Snowball totiž neexistují klasické programovací konstrukce známé z jiných jazyků (`if, else, for` atd.).

Jako další zdroj inspirace jsem použil stemmer pro němčinu. Německý způsob tvoření slov se na rozdíl od angličtiny podobá více českému způsobu. U německého algoritmu už není podrobně vysvětlován přepis do jazyka Snowball, ale i přesto jsem ve zdrojovém kódu našel několik zajímavých konstrukcí.

Další ukázky stemmerů jsem už detailně nezkoumal, protože neovládám gramatiku daných jazyků. Jejich zdrojové kódy, co se použitých konstrukcí týče, vypadaly všechny podobně. Evidentně nejpropracovanějším stemmerem je Porterův anglický. Autor v něm používá nejvíce různých konstrukcí jazyka Snowball. Ostatní stemmery čerpají právě z Porterova nebo z německého stemmeru a jsou méně komplikované.

7.2 Struktura stemmeru pro češtinu

Má implementace stemmeru očekává na vstupu slova skládající se pouze z malých písmen. Kódování vstupních dat musí být UTF-8. Na výstup je převáděn na malá písmena bez diakritiky. Tento převod má na starosti procedura `postlude`, která se volá před koncem programu (po dokončení stemmování). V češtině se při sufixaci objevuje hodně alternací typu *znak s diakritikou* $\rightarrow$ *ekvivalent bez diakritiky* (např. *č* $\rightarrow$ *c*). Toto chování bude odstraněním diakritiky z výstupu ošetřeno a nemusí se kvůli němu zavádět další pravidla.

Kompletní program se skládá ze sekvence deklarací procedur, proměnných a konstant. Následuje sekvence definic procedur. Všechny procedury, které zpracovávají řetězec v obráceném směru (zprava doleva), musí být zahrnuty v deklaraci bloku `backwardmode(...)`. V mé implementaci fungují v tomto obráceném režimu všechny funkce, které odstraňují ze slov sufixy.

API Snowballu umožňuje nadeklarovat proceduru jako externí. Díky tomu je možné získat přístup k této proceduře i v rámci jiného programu. V mém stemmeru je jako externí deklarována procedura `stem`. Tato procedura je volána v demonstračním programu a vrací výsledný stem zadaného slova.

Všechny deklarované proměnné (řetězce, čísla i logické proměnné) jsou dostupné na každém místě programu a existují během célé doby běhu programu. Podobají se statickým deklaracím v jazyce C.

Na začátku zdrojového kódu je nutné uvést seznam non-ASCII znaků vyskytujících se v daném jazyce. Každému znaku je přiřazen zástupný symbol, který se používá v celém zdrojovém kódu. Protože stemmer očekává na vstupu pouze malá písmena, bude seznam obsahovat 15 českých znaků (velká písmena by případně mohla být doplněna obdobně):

```
stringdef a'   hex '0e1'    // á
stringdef cv   hex '10d'    // č
stringdef dv   hex '10f'    // ď
stringdef e'   hex '0e9'    // é
stringdef ev   hex '11b'    // ě
stringdef i'   hex '0ed'    // í
stringdef nv   hex '148'    // ň
stringdef o'   hex '0f3'    // ó
```

```
stringdef rv   hex '159'    // ř
stringdef sv   hex '161'    // š
stringdef tv   hex '165'    // ť
stringdef u'   hex '0fa'    // ú
stringdef u^   hex '16f'    // ů
stringdef y'   hex '0fd'    // ý
stringdef zv   hex '17e'    // ž
```

Následuje definice skupin znaků (*groupings*). Toto seskupení znaků umožňuje odkazovat se na všechny členy skupiny jediným názvem. Tato vlastnost najde uplatnění především při testech. Místo zdlouhavého dotazu typu: "*je na konci řetězce znak A nebo E nebo I. . . ?*"stačí napsat: "*je na konci řetězce samohláska?*". V následujícím příkladu definice skupiny souhlásek a samohlásek je také ukázáno, jak se ve zdrojovém kódu používají zástupné symboly pro znaky s diakritikou:

```
define vowel 'aeiouy{a'}{e'}{ev}{i'}{o'}{u'}{u^}{y'}'
define consonant 'bc{cv}d{dv}fghjklmn{nv}pqr{rv}s{sv}t{tv}vwxz{zv}'
```

Na začátku programu se volá procedura `exception` obsahující různé výjimky a nepravidelně skloňovaná česká slova a jejich stemy. Pokud se zpracovávané slovo nachází v tomto seznamu, vrací program rovnou příslušný stem a nepokouší se aplikovat na slovo žádná pravidla.

Když není slovo na vstupu zachyceno jako výjimka, pokračuje běh programu voláním procedury `mark_regions`. Ta má za úkol vybrat ze slova tzv. úsek R1. Jedná se o úsek začínající za první souhláskou, které předchází samohláska, a pokračující až do konce řetězce.

```
t  r  a  k  t  o  r  i  s  t  a
         |<------------------>|   R1
```

V uvedeném příkladu je písmeno *k* první souhláskou, před kterou stojí samohláska. Úsekem R1 bude tedy řetězec *'torista'*. Má implementace stemmeru provádí veškeré operace se slovy jen nad jejich R1 částmi. Pokud tedy slovo takový úsek vůbec neobsahuje, je ponecháno beze změn a odesláno na výstup. Účelem tohoto opatření je eliminovat zpracování jednoslabičných slov. Jestliže nemá slovo více slabik, pak pravděpodobně nemohlo vzniknout přidáním prefixu nebo sufixu a je tedy zbytečné pokoušet se na něj aplikovat nějaká gramatická pravidla.

Program dále pokračuje voláním procedur, které na základě gramatických pravidel češtiny vytvoří ze vstupního slova jeho stem.

Na stem jsou aplikovány ještě dvě procedury:

1. Korekce zdvojení. Nahrazování koncovek a sufixů může v některých případech způsobit, že na konci stemu se budou vyskytovat dvě stejné souhlásky. Tato procedura jednu z nich odstraní.

2. Korekce samohlásek. Každý stem by měl končit souhláskou. Uvedená procedura toho dosahuje tím, že odstraňuje samohlásky na konci stemu.

7.3 Stemming podstatných jmen

Zásadním zdrojem, ze kterého jsem čerpal informace o gramatických pravidlech týkajících se tvoření slov, byl [5].

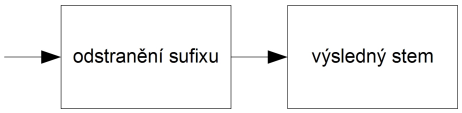
Nejdříve jsem začal zkoumat gramatická pravidla u podstatných jmen, ta totiž obvykle v textu nesou největší množství informace. Tento fakt jsem si ověřil při sestavování testovacích dat pro stemmer. Z náhodně vybraného dostatečně dlouhého novinového článku jsem vybral všechna podstatná jména, přídavná jména a slovesa. Podstatných jmen v něm bylo třikrát více než sloves a dvakrát více než přídavných jmen.

Podstatná jména je možné tvořit odvozováním z jiných podstatných jmen, z přídavných jmen, ze sloves, z číslovek, případně skládáním více slov. Při tvoření slov skládáním dochází k integraci dvou pojmů v jednom pojmenování. Spojovacím článkem obou slov bývá většinou *-o-*. Např. *jihovýchod, lesostep, štěrkopísek...* Skládáním vytvořená slova nemá šanci stemmer rozpoznat (písmeno „o“ sloužící jako spojovník se vyskytuje i ve slovech nesložených), proto jsem se touto skupinou slov dále nezabýval.

Experiment č. 1 - odstraňování sufixů

Podstatná jména jsou zpravidla tvořena sufixací a dají se rozdělit do mnoha skupin, přičemž v každé skupině jsou slova tvořena přidáním různých sufixů. Např. skupina „jmen konatelských“ obsahuje sufixy *-ář, -íř, -ník, -ista, -ák, -ec*. Do skupiny „jmen obyvatelských“ patří sufixy *-an, -ec, -ák* atd. Celkem je ve stemmeru použito 267 různých sufixů.

Jako první experiment jsem tedy použil všechny sufixy uvedené v knize [5]. Činnost stemmeru byla jednoduchá – pokud slovo končilo jedním ze sufixů, došlo k odstranění sufixu a zbývající část slova byla odeslána na výstup (viz obrázek 7.1).



Obrázek 7.1: Schéma experimentu č. 1

Úspěšnost tohoto algoritmu byla podle očekávání velice nízká. Stemmer totiž mohl nalézt suffix pouze u slov v základním tvaru (u podstatných jmen se jedná o 1. pád jednotného čísla). Čeština má ale 7 pádů a jednotné a množné číslo, takže většinu podstatných jmen stemmer nemohl vůbec zpracovat. Výsledky experimentu:

```
správně:      455         (19.6 %)
špatně:      1869 z 2324 (80.4 %)
             under-stemming: 1275
             over-stemming:   575
```

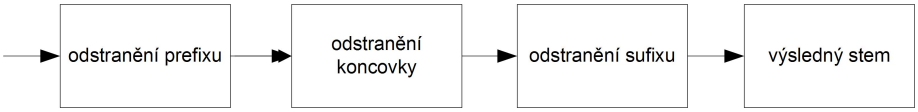
Experiment č. 2 - odstraňování koncovek

Jako další krok jsem se rozhodl odstanit ze slov koncovky vznikající při skloňování podstatných jmen. Koncovky jsem získal ze vzorů podstatných jmen a jejich podtypů. Konkrétně se jednalo o těchto 48 vzorů:

- pán - hoch, občan, demagog, sameček, redaktor
- hrad - ostrov, hříbek
- muž - otec, obyvatel

- stroj - válec
- předseda - husita
- soudce
- žena - barva, jídelna, klika, kočka, pára, volba
- růže - ulice, třešně, Francie
- píseň - mříž, jabloň
- kost - měď
- město - analgetikum, jablko, euro, letadlo, médium, ministerstvo, středisko, okno, synonymum
- moře - bojiště
- kuře - hříbě
- stavení

Snowball ze všech 155 koncovek vybíral automaticky nejdelší, kterou je možné odstranit. Program fungoval tak, že nejdříve zkoušel odstranit koncovky a poté odstraňoval sufixy (stejnou funkcí jako v předchozím experimentu). Schéma činnosti je znázorněno na obrázku 7.2.



Obrázek 7.2: Schéma experimentu č. 2

Výsledky dopadly o několik procent lépe:

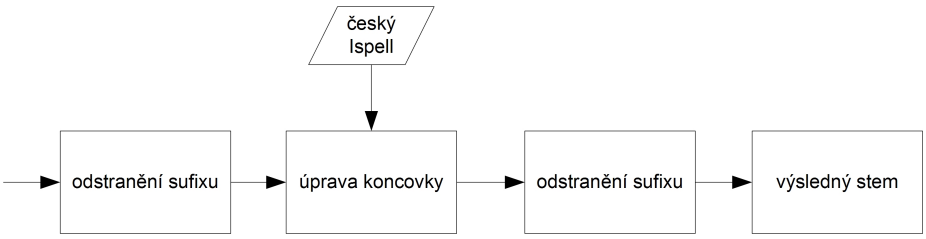
```
správně:      587        (25.3 %)
špatně:      1737 z 2324 (74.7 %)
             under-stemming: 1585
             over-stemming:   121
```

Při tomto experimentu jsem zkoušel z podstatných jmen odstraňovat také prefixy. Podstatná jména sice nejsou tvořena prefixací v takové míře jako slovesa, ale i přesto jich je v [5] uvedeno zhruba 80. Odstraňování prefixů ale vedlo pouze ke zhoršení výsledků, algoritmus nebyl schopný rozlišit, kdy se skutečně jedná o prefix a kdy ne. Např. ve slově *pra-dědeček* je *pra-* opravdu prefixem, stemmer však tento řetězec odstraňoval i ze slov *Praha, prapor* atd. Proto jsem se rozhodl u podstatných jmen prefixy neodstraňovat.

Experiment č. 3 - využití dat z českého Ispellu

V dalším kroku jsem se pokusil pro vylepšení stemmeru využít český slovník pro Ispell[1] (interaktivní program pro kontrolu pravopisu podporující množství evropských jazyků). V tomto slovníku se mj. nachází i přehled českých afixů. Autoři sestavili databázi českých slov a ke každému přidali určité příznaky, které určují, jak se dané slovo v gramatice chová (např. generuje předponu ne-, generuje tvary podle vzoru stroj, generuje tvar s koncovkou -ě/-e pro slova vzoru město atd.). Protože takovéto přiřazování příznaků je velmi zdlouhavé, pokusili se vyvinout skript, který by slova na základě jejich tvaru převáděl do základního tvaru a přiřazoval jim patřičné příznaky. Skript se jim podařilo vyvinout a je distribuován společně s českým slovníkem Ispellu. Nicméně sami autoři poznamenávají, že úspěšnost tohoto skriptu je nízká (kolem 40 %) a výsledky se tedy musí překontrolovávat.

Stemmer však nemusí umět přiřazovat správné příznaky. Stačí, když bude umět převádět slova do základního tvaru, aby bylo snadnější odstranit jejich afixy. Proto jsem se rozhodl zkusit zakomponovat poznatky získané z tohoto skriptu do stemmeru. V tuto chvíli stemmer pracoval tak, že nejdříve odstraňoval sufix, poté zkoušel převést slovo do základního tvaru a následně opět odstraňovat sufix (viz obrázek 7.3).



Obrázek 7.3: Schéma experimentu č. 3

Bohužel dosažené výsledky neodpovídaly mému očekávání.

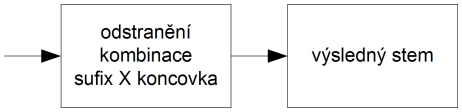
```
správně:    1116        (45.2 %)
špatně:     1198 z 2314 (54.8% )
            under-stemming:  842
            over-stemming:   312
```

Experiment č. 4 - kombinace sufixů s koncovkami

Při skloňování podstatných jmen nedochází k pouhému připojení koncovky za konec slova (např. *les-ník* → *les-ník/ovi*), ale velmi často se vyskytují hláskové alternace (např. zde: *traktor-ista* → *traktor-ist/ovi*; dochází k odstranění písmene „a“). I když tedy stemmer odstraní správnou koncovku, nemůže už odstranit sufix, protože ten už byl odebráním koncovky pozměněn.

Z tohoto důvodu jsem se rozhodl vyzkoušet vygenerovat ke každému sufixu jeho tvary ve všech možných pádech. O každém testovacím slově jsem věděl, k jakému vzoru (případně podtypu) náleží. K sufixu daného slova bylo tedy možné připojit jeho korektní zakončení ve všech pádech (např. vyskloňované sufixy pro slovo *traktor-ista* vypadaly takto: *-ista, -isty, -istovi, -istu, -isto...*). Tímto způsobem jsem získal více než 1 300 spojení sufixů a správného tvaru koncovky. Schéma činnosti stemmeru je znázorněno na obrázku 7.4.

[1] Český slovník pro Ispell je dostupný na adrese `ftp://ftp.tul.cz/pub/unix/ispell/`



Obrázek 7.4: Schéma experimentu č. 4

Výsledky na umělé vytvořené testovací sadě dat byly sice velmi dobré, ale testování s reálnými daty naopak vedlo k velmi nízké úspěšnosti. Tento experiment se tedy neosvědčil.

```
správně:     1961        (84.4 %)
špatně:       363 z 2324 (15.6 %)
             under-stemming:  252
             over-stemming:   109
```

Po bližším prozkoumání OpenOffice slovníku pro kontrolu českého pravopisu[2] jsem zjistil, že je z velké části vybudován na stejném principu – kombinace sufixu se všemi korektními koncovkami. Navíc je však každé pravidlo doplněno o regulární výraz určující, jaké podmínky musí slovo splňovat, aby k němu mohl být daný sufix připojen a jaké případné hláskové alternace je nutné ve slově provést. Protože tento seznam obsahuje také více než tisíc komplikovaných pravidel, rozhodl jsem se jej pro další vývoj nevyužít.

Experiment č. 5 - převádění slov do základního tvaru

Předchozí způsob sice k úspěšnému konci nevedl, ale přesto zůstává převod slov do základního tvaru základním požadavkem pro správnou funkčnost stemmeru. Proto jsem na základě analýzy koncovek získaných v experimentu č. 2 sestavil tabulku mapující koncovky v jednotlivých pádech na jejich tvar v první osobě jednotného čísla. Výsledkem bylo, že 60 % ze všech tvarů koncovek mělo jednoznačně přidělený tvar pro první osobu jednotného čísla. Např. každé slovo zakončené koncovkou *-bou* má v základním tvaru koncovku *-ba* (*oso/bou* → *oso/ba, klen/bou* → *klen/ba* atd.).

U zbývajících 40 % koncovek není možné jednoznačně rozhodnout, jaký základní tvar zvolit. Jednou z koncovek, u kterých toto nelze provést, je např. *-ovi* (starost/ovi → starost/a, ale pán/ovi → pán/0).

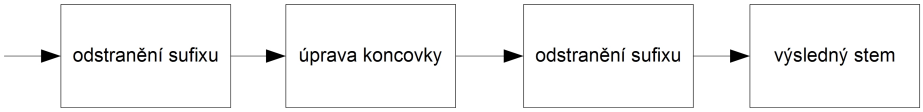
Pokud neznáme kontext zpracovávaného slova, neexistuje žádný způsob založený na gramatických pravidlech, na základě kterého by se dalo rozhodnout o správné koncovce základního tvaru. Proto v mé implementaci stemmeru tyto koncovky pouze mažu. V některých případech to sice může vést k chybnému výsledku, ale pokud bych tyto koncovky ve slově ponechal, budou výsledky chybné v naprosté většině případů.

Program v tomto okamžiku fungoval tak, že se nejprve pokusil odstranit sufix, poté zkoušel převést slovo do základního tvaru a následně opět zkoušel odstranit sufix (viz obrázek 7.5).

Výsledky dopadly o něco hůře, než v experimentu č. 3:

```
správně:     1054        (45.4 %)
špatně:      1270 z 2324 (54.6 %)
             under-stemming: 1012
             over-stemming:   204
```

[2] OpenOffice slovníky jsou dostupné na adrese `http://lingucomponent.openoffice.org/`



Obrázek 7.5: Schéma experimentu č. 5

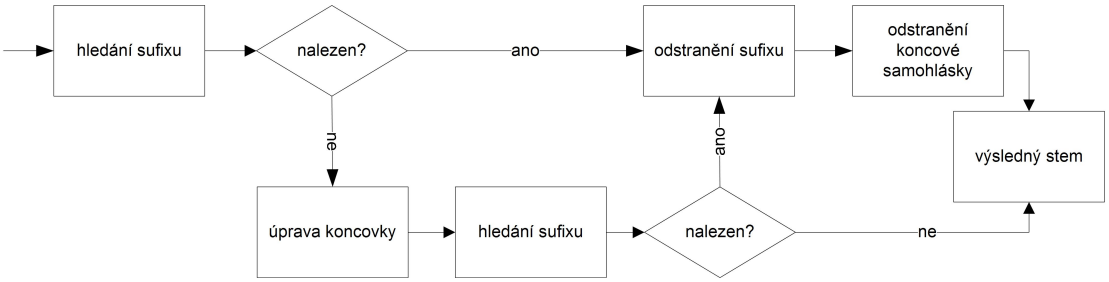
Experiment č. 6 - finální podoba algoritmu

Výsledky předchozího experimentu byly překvapivé, protože nedošlo ke zlepšení výsledků, i když jsem, na rozdíl od experimentu č. 3, který se opíral o data z českého slovníku Ispellu, použil mnohem více pravidel a používal důmyslnější konstrukce nahrazování koncovek. Základní myšlenka algoritmu vypadala správně, proto jsem přistoupil k optimalizaci funkčnosti.

1. Došlo k úpravám pořadí, ve kterém se ze slova odebíraly koncovky. V některých případech docházelo k odstranění pouze části koncovky (např. bylo smazáno *-ím*, ale správně se mělo smazat *-tím* atd.). Dále přestaly být odstraňovány některé jednoznakové koncovky (*-a, -g, -k*...), protože v dalším kroku (odstraňování sufixů) docházelo ke kolizím.

2. K hláskovým alternacím nedochází jen u připojování koncovek ke slovům, ale také při připojování sufixů ke kořenům slov. Proto byla upravena procedura pro odstraňování sufixů. Alternace se doplňovaly hlavně u sufixů začínajících na písmena „i“ nebo „í“ (tzv. alternace typu i).

3. Bylo upraveno pořadí volání procedur. Pokud se hned při prvním vykonání procedury odstraňující sufixy podaří některý sufix nalézt a odstranit, příznak `found_suffix` je nastaven na `true` a už nebude dále volána procedura převádějící slovo do základního tvaru. Ze zbývající části slova se případně odstraní koncová samohláska a výsledný stem bude poslán na výstup (viz obrázek 7.6).



Obrázek 7.6: Schéma experimentu č. 6

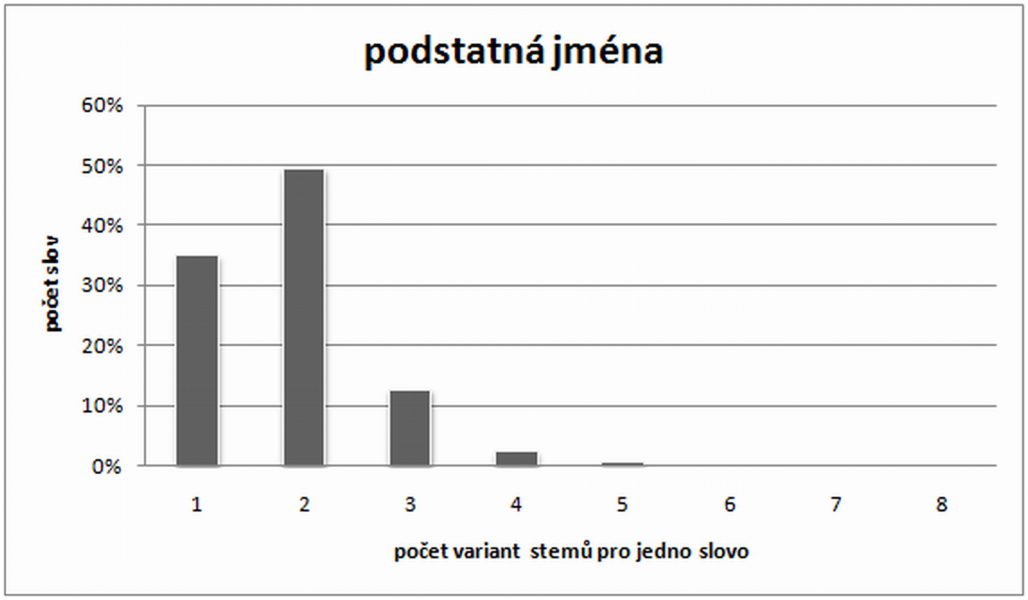
Provedení všech těchto úprav skutečně vedlo ze zlepšení výsledků stemmeru. Výsledky testování na uměle vytvořené sadě dat:

```
správně:     1433        (61.9 %)
špatně:       881 z 2314 (38.1 %)
             under-stemming:  663
             over-stemming:   171
```

Výsledky testování na datech vytvořených z novinových článků:

```
správně:     223       (74.6 %)
špatně:       76 z 299 (25.4 %)
             under-stemming:  27
             over-stemming:   49
```

V testovací sadě podstatných jmen existovalo každé slovo v průměrně 8,6 variantách. Obrázek 7.7 ukazuje, kolik procent slov bylo převedeno na určitý počet různých stemů. Z obrázku je patrné, že u 80 % slov se vyskytnuly maximálně dva různé stemy. Analýzou výsledků jsem navíc došel k závěru, že u většiny slov převedených na dva různé stemy se jednalo o důsledek stejného gramatického pravidla, které stemmer nebyl schopný zpracovat.



Obrázek 7.7: Výsledky stemmování podstatných jmen.

Tato varianta stemmeru se ukázala z hlediska poskytovaných výsleků jako nejlepší, takže jsem se rozhodl využít ji i u stemmování dalších slovních druhů.

7.4 Stemming přídavných jmen

Po dokončení stemmeru podstatných jmen jsem pokračoval přídavnými jmény, protože v textech se objevují častěji než slovesa. Poněvadž přídavná jména jsou stejně jako jména podstatná tvořena hlavně sufixací, rozhodl jsem se vyzkoušet pro jejich stemování obdobný princip jako pro podstatná jména.

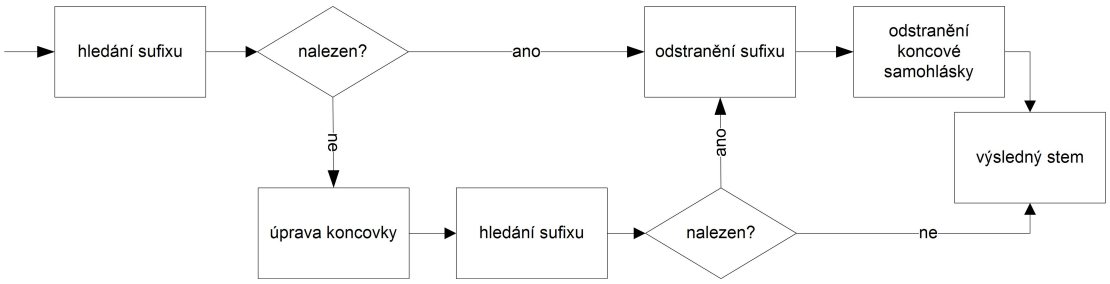
Přídavná jména je možné tvořit odvozováním z podstatných jmen, ze sloves, z jiných přídavných jmen, případně skládáním. Podobně jak u podstatných jmen dochází u tvoření skládáním ke spojení konektem *-o-*. Nejčastěji se spojuje adjektivní základ se substantivním. Ani u přidávných jmen jsem tímto způsobem vytvořená slova do stemmeru nezahrnul.

Má implementace stemmeru pracuje se 101 různými sufixy přídavných jmen. Na rozdíl od situace u podstatných jmen zde dochází k mnohem častějším alternacím. Převládají alternace $h \rightarrow ž$, $ch \rightarrow š$, $k \rightarrow č$.

Koncovky přídavných jmen byly získány zanalyzováním následujících 14 vzorů a jejich podtypů: bystrý, bývalý, český, drahý, hladký, jarní, levý, matčin, mladý, otcův, starý, suchý, vesnický, vysoký.

Z těchto vzorů jsem získal celkem 126 různých koncovek. Na rozdíl od situace u podstatných jmen se zde podařilo u většiny slov podle jejich koncovky určit, jakou podobu má koncovka pro základní tvar (1. pád mužského rodu).

Stemmer pracuje tak, že nejdříve zkusí odstranit sufix. Pokud žádný nenajde, pokouší se převést slovo do základního tvaru a poté znovu hledá sufix, který by mohl odstranit (viz obrázek 7.8).



Obrázek 7.8: Schéma experimentu s přídavnými jmény.

U přídavných jmen jsem zkoušel stejné varianty běhu programu jako u podstatných jmen, ale tato dávala nejlepší výsledky. Výsledky testování na uměle vytvořené sadě dat:

```
správně:    595       (82.5 %)
špatně:     126 z 721 (17.5 %)
            under-stemming:   89
            over-stemming:    35
```

Výsledky testování na datech vytvořených z novinových článků:

```
správně:    61       (45.9 %)
špatně:     72 z 133 (54.1 %)
            under-stemming:  9
            over-stemming:  62
```

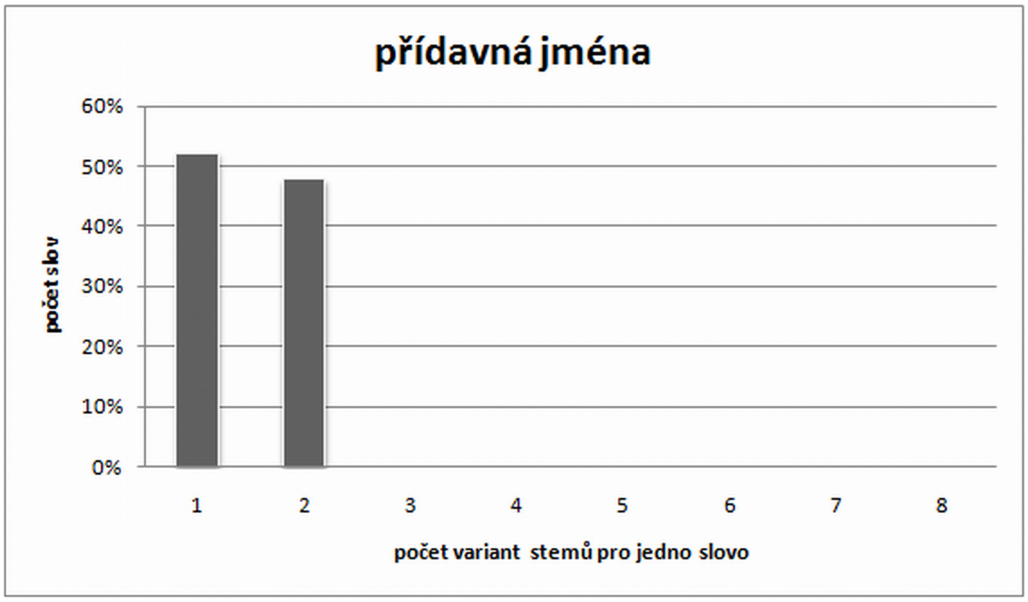
V testovací sadě přídavných jmen existovalo každé slovo v průměrně 10,8 variantách. Obrázek 7.9 ukazuje, kolik procent slov bylo převedeno na určitý počet různých stemů.

Stemmování přídavných jmén dopadlo ze všech slovních druhů nejlépe – každé slovo bylo převedeno na maximálně dva různé stemy.

7.5 Stemming sloves

Způsob tvorby sloves se liší od postupů platných pro podstatná a přídavná jména. Při sufixaci se u víceslabičných sloves vychází z podoby jejich infinitivního kmene, proto se pokouším slovesa převádět do infinitivu. Mnohem větší roli než u ostatních slovních druhů hraje při vytváření sloves prefixace. Důležitým poznatkem je, že hláskové alternace jsou velmi vzácné.

Stemmer zpracovává celkem 15 sufixů a 29 prefixů. Při převodu slovesa na infinitiv se pracuje s 345 koncovkami, které byly získány z těchto 18 vzorů: brát, čistit, chodit, krýt,

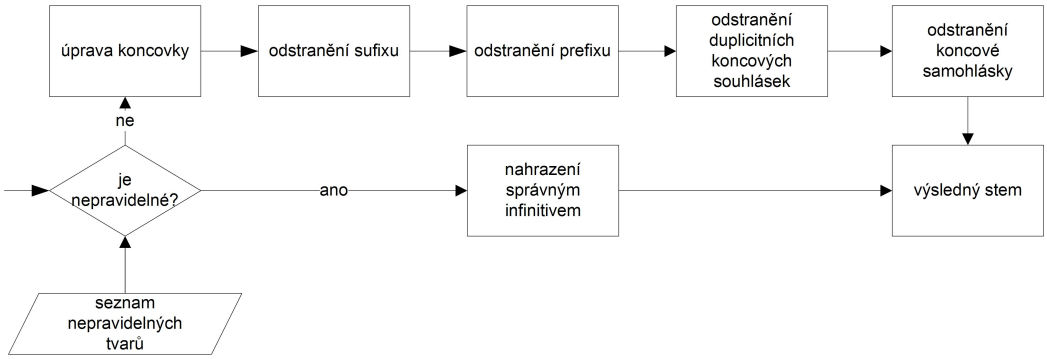


Obrázek 7.9: Výsledky stemmování přídavných jmen.

kupovat, mazat, minout, nést, péct, prosit, rušit, sázet, tisknout, tnout, trpět, třít, volat, vozit.

U většiny koncovek se podařilo jednoznačně určit, jak podle nich převést sloveso do infinitivu. Čeština však obsahuje i 8 nepravidelných sloves (*být, mít, jít, jíst, sníst, vidět, vědět, chtít*). S nimi by si stemmer na základě gramatických pravidel neporadil, proto je přímo ve zdrojovém kódu všem tvarům těchto sloves přiřazen správný infinitivní tvar.

Opět jsem vyzkoušel stejný princip algoritmu, který se osvědčil již u podstatných a přídavných jmen. U sloves ale nedosahoval tak dobrých výsledků. První volání procedury hledající sufixy velmi často odstranilo nesprávný úsek slova. Proto jsem po několika testech dospěl k variantě, kdy se nejdříve na základě koncovky převádí sloveso do základního tvaru a až poté se odstraňuje sufix. Dále se odstraňuje případný prefix. Protože na konci výsledného stemu často docházelo ke zdvojení některých souhlásek, přidal jsem ještě proceduru, která tyto duplicity odstraní. Schéma činnosti je znázorněno na obrázku 7.10.



Obrázek 7.10: Schéma průběhu stemmování sloves.

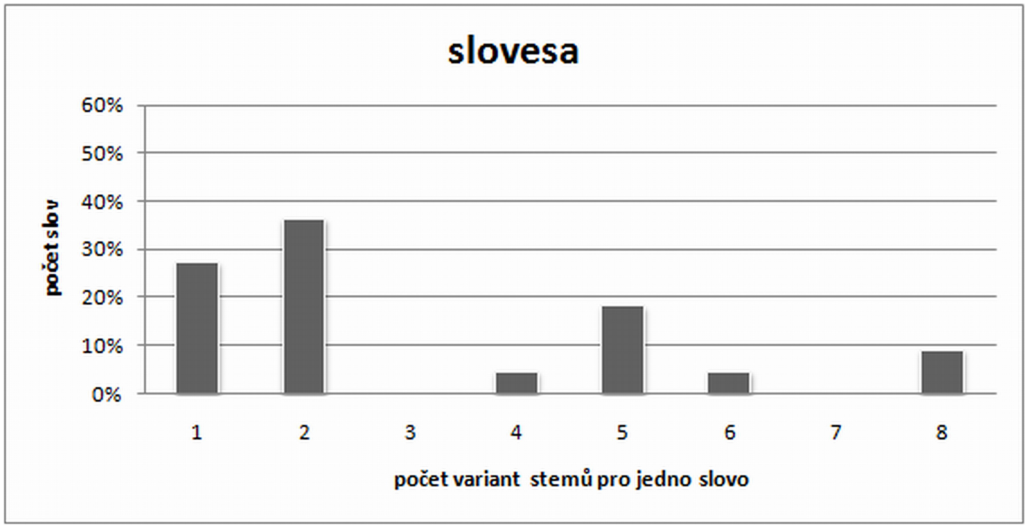
Algoritmus uvedený na obrázku 7.10 poskytoval na uměle vytvořené sadě dat tyto výsledky:

```
správně:     347       (70.5 %)
špatně:      145 z 492 (29.5 %)
             under-stemming:   35
             over-stemming:    76
```

Výsledky testování na datech vytvořených z novinových článků:

```
správně:     66       (46.8 %)
špatně:      75 z 141 (53.2 %)
             under-stemming:  35
             over-stemming:   18
```

V testovací sadě sloves existovalo každé slovo průměrně v 23,4 variantách. Obrázek 7.11 ukazuje, kolik procent slov bylo převedeno na určitý počet různých stemů.



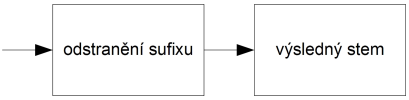
Obrázek 7.11: Výsledky stemmování sloves.

Z obrázku je patrné, že u sloves dopadly výsledky hůře, než v předchozích případech. V porovnání s podstatnými jmény poklesla úspěšnost stemmování sloves přibližně o 30 %. Navíc významně narostl počet slov, která jsou převedena na pět a více různých stemů. Tento fakt je zapříčiněn tím, že v testovací sadě se slovesa vyskytují v mnohem více tvarech než předchozí slovní druhy. Dalším důvodem může být skutečnost, že testovací sada obsahuje některé v češtině málo používané tvary sloves (např. přechodníky), se kterými si stemmer neporadí. Navíc i předchozí testy ukazovaly u sloves horší výsledky.

7.6 Stemming příslovcí

Slovotvorba příslovcí nemá taková složitá pravidla jako předchozí slovní druhy. Mohou být tvořena z podstatných a přídavných jmen, sloves, příslovcí, zájmen a číslovek. Vznikat mohou prefixací i sufixací. Odstraňování prefixů jsem do stemmeru nezahrnul, protože často

docházelo k chybným smazáním řetězce, který ve skutečnosti prefixem nebyl. Některé ze sufixů jsou zachyceny už při zpracování jiných slovních druhů, takže u příslovcí jich stemmer odstraňuje pouze 6: *ky, mo, si, koli, koliv, hle*. Schéma činnosti stemeru pro příslovce je tedy jednoduché – odstraňuje pouze sufixy (viz obrázek 7.12).



Obrázek 7.12: Schéma průběhu stemmování příslovcí.

Protože v porovnání s ostatními slovními druhy není u příslovcí takové množství sufixů a gramatických pravidel pro připojení sufixu ke kořenu slova, nesestavoval jsem pro ně testovací sadu dat, na které by mohlo být prováděno testování.

7.7 Algoritmus pro stemmování češtiny

Po vytvoření stemmerů pro jednotlivé slovní druhy bylo nutné je spojit do jednoho celku, který by zvládal správně zpracovat slova všech slovních druhů dohromady. Protože stemmer nemá vstupní slovo zařazeno do kontextu, nezná o něm tedy žádné podrobnější informace (např. o jaký slovní druh se jedná, v jakém je čísle atd.) a nemůže tak rozhodnout, který stemmovací algoritmus použít. Proto jsem pořadí zpracování určil na základě četnosti výskytu jednotlivých slovních druhů v českých textech.

Nejdříve dochází ke stemmování podstatných jmen. Pokud se nepodaří určit stem, následují přídavná jmena a příslovce. Slovesa jsou zpracována jako poslední. Mohla by sice přijít na řadu ještě před příslovci, ale stemmer sloves se během testování jevil jako agresivní – zpracovával i jiné slovní druhy, než měl. Navíc je u citoslovcí odstraňován jen malý počet sufixů, takže by zařazení citoslovcí před slovesa nemělo negativně ovlivnit výsledky stemmeru.

Krátká slova (většinou jednoslabičná) nemohou být už z principu vytvořena prefixací, sufikací nebo skládáním slov. Je tedy vhodné, aby stemmer s těmito slovy nepracoval a neodstraňoval z nich žádné části. V mé implementaci stemmer pracuje pouze se slovy delšími než 4 znaky. Hodnota 4 byla zjištěna experimentálně – docházelo s ní k nejmenšímu počtu chyb.

Na závěr stemmer ze slova odstraňuje případné koncové duplicitní souhlásky, případné koncové samohlásky a z výsledného stemu odstraní diakritiku.

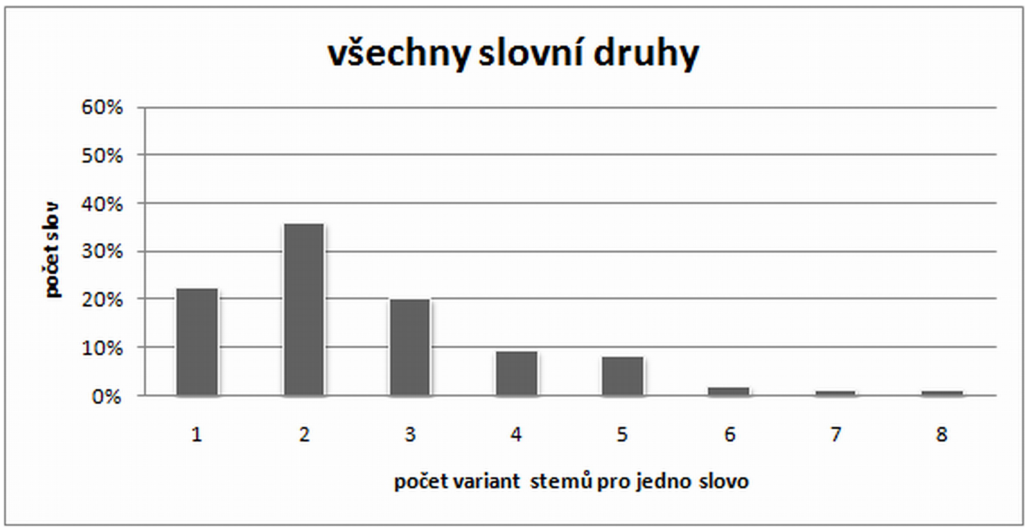
Kompletní princip činnosti stemmeru je detailně znázorněn na obrázku 7.14 a jeho výsledky na uměle vytvořené sadě dat jsou následující:

```
správně:    2065        (58.5 %)
špatně:     1462 z 3527 (41.5 %)
            under-stemming:  746
            over-stemming:   588
```

Výsledky testování na datech vytvořených z novinových článků:

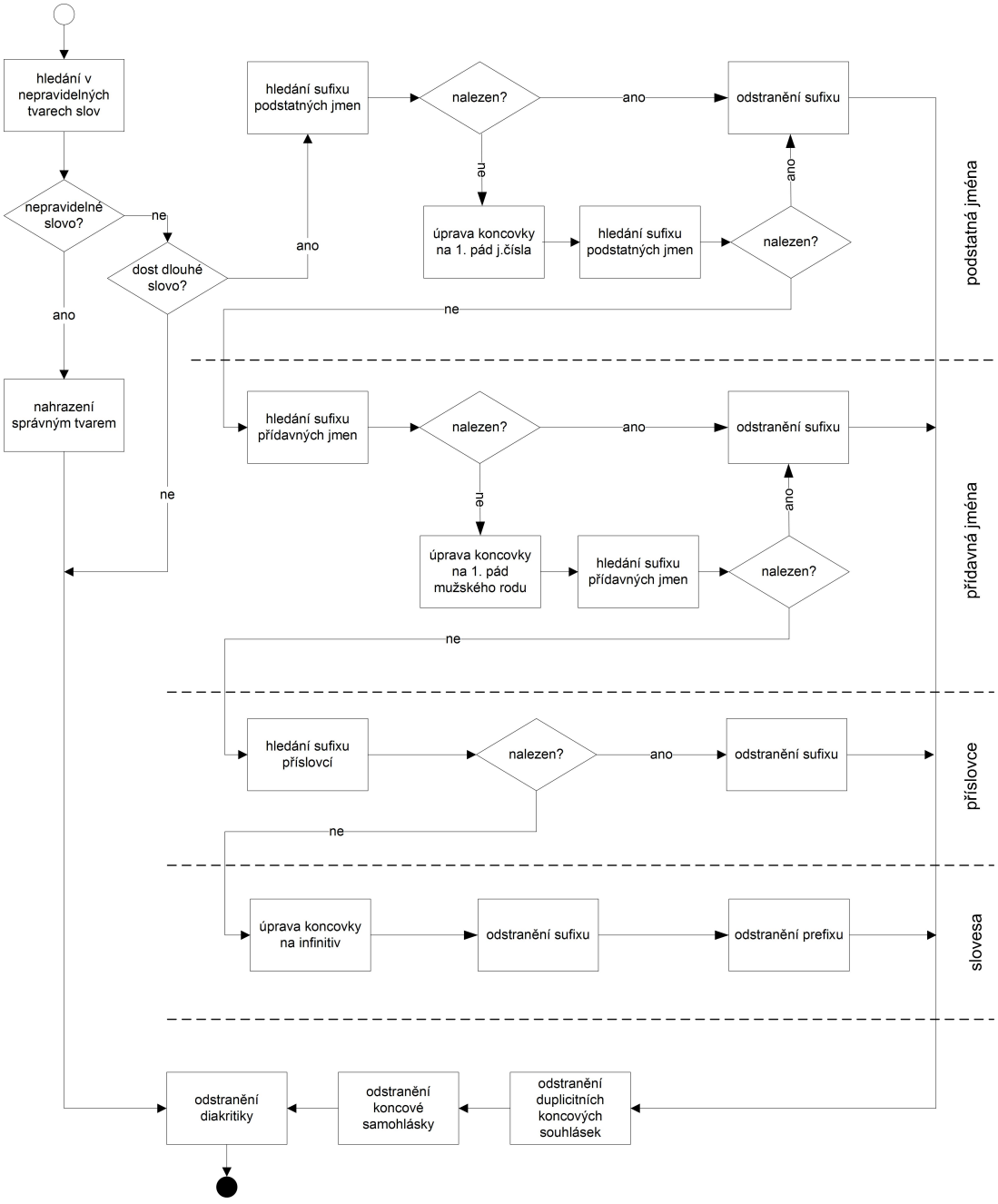
```
správně:    276       (48.2 %)
špatně:     297 z 573 (51.8 %)
            under-stemming:  108
            over-stemming:   155
```

V testovací sadě všech slovních druhů pohromadě existovalo každé slovo průměrně v 9,8 variantách. Obrázek 7.13 ukazuje, kolik procent slov bylo převedeno na určitý počet různých stemů. Z obrázku je patrné, že se více než 50 % slov převede na maximálně dva různé stemy. Statistiku zhoršují především výsledky stemmování sloves a fakt, že některá slova byla stemmována podle pravidel pro jiný slovní druh.



Obrázek 7.13: Výsledky stemmování všech slovních druhů.

Výsledná podoba stemmeru byla testována na kombinaci všech předchozích testovacích dat. Vznikl tak soubor podstatných jmen, přídavných jmen a sloves o celkové velikosti 3 530 slov. Podle očekávání se správnost výsledku snížila, protože u některých slov docházelo ke zpracování špatnou části stemmeru (např. s přídavným jménem mohlo být zacházeno jako se slovesem apod.). Těmto situacím se však dá bez doplňujících informací o slově jen těžce zabránit. Ukázka poskytovaných výsledků je v příloze B.



Obrázek 7.14: Schéma činnosti stemmeru češtiny.

Kapitola 8

Závěr

Cílem této diplomové práce byl vývoj lemmatizačního algoritmu založeného na gramatických pravidlech pro český jazyk. Implementace měla proběhnout v jazyce Snowball, což je speciální jazyk pro popis lemmatizačních algoritmů. Zadání se podařilo úspěšně splnit.

Během vývoje algoritmu neexistoval jiný volně dostupný lemmatizátor pracující na stejném principu. Pro český jazyk existují slovníkové lemmatizátory založené zpravidla na datech z českého slovníku pro Ispell a také statistický lemmatizátor indických autorů založený na vzdálenostních funkcích.

Testování nově vyvinutého lemmatizátoru na reálných datech získaných z novinových článků ukázalo, že zhruba v 50 % případů nalezne správný gramatický kořen slova. U uměle sestavených testovacích dat snažících se pokrýt co možná nejširší část gramatických pravidel nalezne správný kořen slova přibližně v 60 % případů.

Dalším kritériem pro testování byl počet tvarů, na které lemmatizátor převede všechny existující varianty daného slova. V tomto případě probíhalo testování pouze na uměle vygenerovaném seznamu slov. Téměř 80 % slov, která se vyskytovala průměrně v 9,8 různých variantách, se podařilo převést na maximálně 3 různé stemy. Výsledky na reálných datech by mohly být ještě úspěšnější, protože testovací data záměrně obsahovala množství výjimek a málo používaných tvarů slov.

Jako zdroj gramatických pravidel, na jejichž základě lemmatizátor pracuje, sloužil [5]. Během vývoje byla použita také některá data z českého slovníku pro Ispell: soubor stopslov pro češtinu zčásti vychází ze seznamu zájmen, spojek a citoslovcí v Ispellu a dalším využitým prvkem jsou tvary některých nepravidelných slov.

Finální implementace algoritmu nyní pracuje jako pět za sebou následujících samostatných částí (zpracování nepravidelných slov, moduly pro podstatná jména, přídavná jména, příslovce a slovesa). Pořadí volání jednotlivých modulů je odvozeno na základě četnosti výskytu jednotlivých slovních druhů v češtině. Přesto v některých případech může dojít k tomu, že slovo bude zpracováno modulem pro jiný slovní druh, než by mělo být.

Řešením tohoto problému by mohlo být spojení všech modulů do jednoho. Tento způsob však komplikuje obrovské množství prefixů, sufixů a koncovek slov, se kterými lemmatizátor pracuje. Bylo by nutné provádět velké množství experimentů se seřazením jednotlivých pravidel tak, aby byla všechna aplikována v takovém pořadí, že mezi nimi nebude docházet ke kolizím.

Dalším vylepšením stávajícího algoritmu by mohlo být zakomponování některých pravidel z OpenOffice slovníku pro kontrolu pravopisu. Postavit algoritmus výhradně na těchto pravidlech by pravděpodobně nebylo dobrým řešením, protože jsou založeny na kombinacích sufixů se všemi odpovídajícími koncovkami. Podobný experiment byl během vývoje

algoritmu proveden a výsledky nebyly dostatečně kvalitní. V OpenOffice slovníku jsou ale jednotlivá pravidla doplněna o regulární výrazy, které určují, kdy se může dané pravidlo aplikovat. Této skutečnosti by mohlo být využito pro ošetření některých chyb produkovaných stávajícím algoritmem.

Největším problémem pro správné fungování lemmatizátoru založeného na gramatických pravidlech je fakt, že nezná kontext zpracovávaného slova, tudíž o něm nemá žádné doplňující informace (např. o jaký slovní druh se jedná atd.). Dalším palčivým problémem jsou některé hláskové alternace. Např. u sufixů začínajících písmenem „i“ se může koncové „ž“ ve slově měnit buď na „g“, nebo na „h“.

Na základě těchto problémů a obecně složitosti české gramatiky by se mi jako vhodný pro češtinu jevil hybridní lemmatizátor – byl by také založen na gramatických pravidlech, ale zároveň by měl k dispozici databázi s českou slovní zásobou. Pokud by během převádění slova na jeho základní tvar narazil na nějakou nejednoznačnou situaci, mohl by si v databázi ověřit, jestli po aplikování daného pravidla získá existující české slovo. Pokud by zjistil, že nezíská, pokračoval by v aplikování dalších pravidel. V databázi by tedy mohla být uložena pouze slova v jejich základních tvarech a jako její zdroj by mohly sloužit volně dostupné slovníky pro kontrolu pravopisu (český Ispell, OpenOffice).

Toto schéma ovšem nelze v jazyce Snowball vytvořit. Není totiž možné využívat připojení k databázi, takže by se celý slovník musel nacházet ve zdrojovém kódu programu, což by mělo negativní vliv na jeho velikost a rychlost prováděné lemmatizace. Řešením by bylo rozšířit jazyk Snowball o přístup k databázím.

Literatura

[1] BAEZA-YATES, R.; RIBEIRO-NETO, B.: *Modern Information Retrieval*. Addison Wesley, 1999, ISBN 020139829X.

[2] BARTUNOV, O.; SIGAEV, T.: *Tsearch2 - full text extension for PostgreSQL*. [online], [cit. 2010-01-01].
URL `<http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/>`

[3] JANDA, L. A.; TOWNSED, C. E.: *Czech Reference Grammar*. [online], [cit. 2010-01-01].
URL `<http://www.seelrc.org:8080/grammar/mainframe.jsp?nLanguageID=2>`

[4] KARLÍK, P.; NEKULA, M.; PLESKALOVÁ, J.: *Encyklopedický slovník češtiny*. Nakladatelství Lidové noviny, 2002, ISBN 80-7106484-X, 604 s.

[5] Kolektiv autorů: *Příruční mluvnice češtiny*. Nakladatelství Lidové noviny, 1995, ISBN 80-7106-134-4, 799 s.

[6] Kolektiv autorů: *Pravidla českého pravopisu*. Academia, 2005, ISBN 80-200-1327-X.

[7] LOVINS, J. B.: *Development of a Stemming Algorithm**. [online], [cit. 2010-01-01].
URL `<http://www.mt-archive.info/MT-1968-Lovins.pdf>`

[8] MAJUMDER, P.; MITRA, M.; PAL, D.: *Hungarian and Czech Stemming using YASS*. [online], [cit. 2010-01-01].
URL
`<http://www.clef-campaign.org/2007/working_notes/majumderCLEF2007.pdf>`

[9] PORTER, M. F.: *Snowball: A language for stemming algorithms*. [online], [cit. 2010-01-01].
URL `<http://snowball.tartarus.org/texts/introduction.html>`

[10] PORTER, M. F.: *An algorithm for suffix stripping. Program*, 1980.

[11] STĚHULE, P.: *Fulltextování v PostgreSQL - modul tsearch2*. [online], [cit. 2010-01-01].
URL
`<http://www.root.cz/clanky/fulltextovani-v-postgresql-modul-tsearch2/>`

[12] WIKIPEDIA: *Stemming. Wikipedia*. [online], [cit. 2010-01-01].
URL `<http://en.wikipedia.org/wiki/Stemming>`

[13] ČECHOVÁ, M.; aj.: *Možnosti a meze české gramatiky*. Academia, 2000, ISBN 80-85866-57-9.

Příloha A

Obsah přiloženého CD

Adresářová struktura přiloženého CD je následující:

- `text`
  - `dp_hellebrand.pdf` – tato diplomová práce ve formátu pdf
  - `src` – zdrojové soubory textu této diplomové práce v systému LaTeX
- `program`
  - `examples` – zdrojové kódy demonstračního programu umožňujícího spouštět stemmery vytvořené v jazyce Snowball
  - `inputs` – textové soubory sloužící jako vstupní data pro stemmer
  - `libstemmer` – zdrojové soubory zpřístupňující základní stemmovací funkcionalitu (*libstemmer.c*) a definice jazyků, s kterými bude možné pracovat (*modules.h*)
  - `runtime` – API jazyka Snowball
  - `scripts` – pomocné skripty v jazyce Perl
  - `snowball` – zkompilovaný program Snowball, skripty v jazyce Snowball (soubory .sbl), seznam českých stopslov
  - `src_c` – zde ukládá program Snowball skripty přeložené do jazyka C
  - soubor `Makefile` umožňuje snadné vytvoření demonstračního programu

Příloha B

Ukázka výstupu programu

tramvajáka	tramvaj	pravičáka	pravic
tramvaják	tramvaj	pravičák	pravic
tramvajákem	tramvaj	pravičákem	pravic
tramvajácích	tramvaj	pravičácích	pravic
tramvajáky	tramvaj	pravičáky	pravic
tramvajáku	tramvaj	pravičáku	pravic
tramvajákovi	tramvaj	pravičákovi	pravic
tramvajákům	tramvaj	pravičákům	pravic
tramvajáků	tramvaj	pravičáků	pravic
tramvajáci	tramvaj	pravičáci	pravic
metodika	metod	schodištím	schod
metodik	metod	schodišti	schod
metodikem	metod	schodiště	schod
metodicích	metod	schodištěm	schod
metodiky	metod	schodištích	schod
metodiku	metod	schodišť	schod
metodikovi	metod	zálesáka	zales
metodikům	metod	zálesák	zales
metodiků	metod	zálesákem	zales
metodici	metod	zálesácích	zales
psycholozi	psych	zálesáky	zales
psychologů	psych	zálesáku	zales
psychology	psych	zálesákovi	zales
psychologovi	psych	zálesákům	zales
psychologové	psych	zálesáků	zales
psychologa	psych	zálesáci	zales
psychologům	psych	jeřábník	jerab
psychologem	psych	jeřábníkem	jerab
psycholog	psych	jeřábníky	jerab
psycholozích	psych	jeřábnících	jerab
psychologu	psych	jeřábníku	jerab
sochařem	soch	jeřábníkovi	jerabn
sochařích	soch	jeřábníkům	jerab
sochaře	soch	jeřábníci	jerab
sochař	soch	jeřábníků	jerab

sochařům	soch	boxere	box
sochařů	sochar	boxerové	box
házenkářem	hazen	boxera	box
házenkářích	hazen	boxer	box
házenkáře	hazenk	boxerech	box
házenkář	hazen	boxerem	box
házenkářům	hazen	boxery	box
házenkářů	hazenkar	boxeru	box
házenkáři	hazenk	boxerovi	box
bankéřem	bank	boxerům	box
bankéřích	bank	boxerů	boxer
bankéře	bank	boxeři	box
bankéř	bank	římane	rim
bankéřům	bank	římana	rim
bankéřů	banker	říman	rim
bankéři	bank	římanech	rim
vousatému	vous	římanem	rim
vousatou	vous	římany	rim
vousatými	vous	římanu	rim
vousatá	vous	římanovi	rim
vousatější	vous	sírovému	sir
vousatým	vous	sírovou	sir
vousatý	vous	sírovými	sir
vousatí	vous	sírová	sir
vousatém	vous	sírovým	sir
vousatých	vous	sírový	sir
vousatého	vous	síroví	sir
vousaté	vous	sírovém	sir
vyprahlému	vyprah	sírových	sir
vyprahlejší	vyprah	sírového	sir
vyprahlou	vyprah	sírové	sir
vyprahlými	vyprah	stojatému	stoj
vyprahlá	vyprah	stojatému	stoj
vyprahlým	vyprah	stojatou	stoj
vyprahlý	vyprah	stojatými	stoj
vyprahlí	vyprah	stojatá	stoj
vyprahlém	vyprah	stojatější	stoj
vyprahlých	vyprah	stojatým	stoj
vyprahlého	vyprah	stojatý	stoj
vyprahlé	vyprah	stojatí	stoj
proste	prosit	stojatém	stoj
prosím	prosit	stojatých	stoj
prosí	pros	stojatého	stoj
prošeno	pros	stojaté	stoj
prosme	prosit	mladičkého	mlad
prosil	prosit	mladičkou	mlad
prosíc	prosit	mladičké	mlad
prošen	pros	mladičcí	mladic

prošeni	pros	mladičkému	mlad
prosily	pros	mladičkých	mlad
prosíce	pros	mladičkém	mlad
prosilo	pros	mladičká	mlad
prošena	pros	mladičký	mlad
prošeny	pros	mladičkými	mlad
pros	prosit	mladičkým	mlad
prosíš	prosit	vozil	voz
prose	prosit	vozily	voz
prosíte	prosit	voženy	vozat
prosíme	prosit	voženi	vozat
prosila	pros	vozme	voz
tiskněme	tisknout	vozíš	voz
tiskněte	tiskn	voženo	voz
tisknu	tisk	vozili	voz
tiskneme	tisknout	vožena	voz
tisknete	tisknout	vozíc	voz
tištěn	tisten	voze	voz
tištěna	tist	vozím	voz
tiskli	tisknout	vozte	voz
tisknut	tisknout	vozila	voz
tiskla	tisk	vozíce	voz
tiskneš	tisknout	vozíme	voz
tisknuta	tisknout	vozí	voz
tisknuty	tisknout	voz	voz
tisknuti	tisknut	vožen	vozat
tištěni	tist	vozilo	voz
tiskna	tisk	vozíte	voz
tiskly	tisk		
tiskl	tisknout		
tiskni	tisk		
tištěno	tist		
tisknouc	tisknout		
tisknouce	tiskn		
tištěny	tist		
tisklo	tisk		
tisknou	tisk		
tiskne	tisknout		
tisknuto	tisknout		

Příloha C

Přehled pomocných skriptů

V adresáři `scripts` na přiloženém CD se nachází následující skripty v jazyce Perl:

- `afixes.pl`
  Převádí seznamy slov do podoby vyžadované jazykem Snowball.
- `make_test_sada_podstjm.pl`
  Vygenerování všech existujících variant podstatných jmen.
- `make_test_sada_pridjm.pl`
  Vygenerování všech existujících variant přídavných jmen.
- `make_test_sada_slovesa.pl`
  Vygenerování všech existujících variant sloves.
- `results.pl`
  Zjištění počtu správných, částečně správných a chybných výsledků.
- `results2.pl`
  Zjištění počtu různých stemů, na které se převedou všechny varianty jednoho slova.

Detailní popisy činnosti jsou uvedeny přímo v jednotlivých skriptech.